CRANFIELD UNIVERSITY

HUGO COURTOIS

OBSTACLE AVOIDANCE FOR UNMANNED AERIAL VEHICLES DURING TELEOPERATION

SCHOOL OF DEFENCE AND SECURITY
Electronic Warfare Information and Cyber

PhD
Academic Year: 2015–2019

Supervisor: Prof. Nabil Aouf
September 2019

CRANFIELD UNIVERSITY

SCHOOL OF DEFENCE AND SECURITY
Electronic Warfare Information and Cyber

PhD

Academic Year: 2015–2019

HUGO COURTOIS

Obstacle Avoidance for Unmanned Aerial Vehicles During Teleoperation

Supervisor: Prof. Nabil Aouf
September 2019

Abstract

Unmanned Aerial Vehicles (UAVs) use is on the rise, both for civilian and military applications. Autonomous UAV navigation is an active research topic, but human operators still provide a flexibility that currently matches or outperforms computers controlled aerial vehicles. For this reason, the remote control of a UAV by a human operator, or teleoperation, is an important subject of study. The challenge for UAV teleoperation comes from the loss of sensory information available for the operator who has to rely on onboard sensors to perceive the environment and the state of the UAV. Navigation in cluttered environment or small spaces is especially hard and demanding. A flight assistance framework could then bring significant benefits to the operator.

In this thesis, an intelligent flight assistance framework for the teleoperation of rotary wings UAVs in small spaces is designed. A 3D haptic device serves as a remote control to improve ease of UAV manipulation for the operator. Moreover, the designed system provides benefits regarding three essential criteria: safety of the UAV, efficiency of the teleoperation and workload of the operator. In order to leverage the use of a 3D haptic controller, the initial obstacle avoidance algorithm proposed in this thesis is based on haptic feedback, where the feedback repels the UAV away from obstacles. This method is tested by human subjects, showing safety benefits but no manoeuvrability improvements. In order to improve on those criteria, the perception of the environment is studied using Light Detection And Ranging (LIDAR) and stereo cameras sensors data. The result of this led to the development of a mobile map of the obstacles surrounding the UAV using the LIDAR in addition to the stereo camera adopted to improve density. This map allows the creation of a flight assistance system that analyses and corrects the user's inputs so that collisions are avoided while improving manoeuvrability. The proposed flight assistance system is validated through experiments involving untrained human subjects in a synthetically simulated environment. The results show that the proposed flight assistance system sharply reduces the number of collisions, the time required to complete the navigation task and the workload of the participants.

Keywords

UAV, teleoperation, obstacle avoidance, mapping, Normal Distribution Transform, LIDAR, haptic.

À mes parents et mon frère

Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

List of Abbreviations

ANOVA	ANalysis Of VAriance
CPU	Central Processing Unit
D2D NDT	Distribution to Distribution NDT
ELAS	Efficient Large Scale Stereo Matching
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
NDT	Normal Distribution Transform
NDT RC	Normal Distribution Transform ReCentring
OAST	Obstacle Avoidance System for Teleoperation
PRF	Parametric Risk Field
RMSE	Root Mean Square Error
RSC	Recursive Sample Covariance
SGM	Semi Global Matching
SLAM	Simultaneous Localisation And Mapping
TLX	Task Load Index
UAV	Unmanned Aerial Vehicle

Acknowledgements

I am extremely grateful to my supervisor, Prof. Nabil Aouf, for giving me the opportunity to undertake this PhD. His support and knowledge throughout this project have been invaluable and I am grateful for his confidence in my work, which would not have been possible without him.

I would like to show my appreciation to Dr. David James for accepting to supervise me during the end of my PhD and providing useful suggestions.

I express my sincere gratitude to Dr. Kenan Ahiska and Dr. Marco Cecotti for the invaluable help they provided me during the last year of my thesis. Our discussions have helped me grow as a scientist and the numerous comments on the thesis have been a tremendous aid.

I would like to thank the staff and students of the lab for the interesting discussions we had. A special mention to Axel, Alejandro, Ozgun, Luc and Duarte for their friendship and the great moments we spent together.

I express my heartfelt thanks to my parents and my brother, who not only brought me support during this PhD, but helped me grow as a person during my whole life.

I would like to thank Carole Belloni in particular, for making my life more interesting and enjoyable during the PhD journey.

I would like to thank Dr. Trevor Ringrose for his help and insights on statistical analysis.

I would like to thank the EPSRC for providing funding for this research.

Chapter 1

Introduction

Contents

1.1 Context

The technological development of Unmanned Aerial Vehicles (UAVs) results in challenges that are both technical and regulatory [7]. While UAVs are historically used in military applications, they are increasingly adopted for civilian purposes. A study conducted for the European Commission in 2007 [8] identifies several key application markets in the civilian domain: government, fire fighting, energy sector, agriculture, forestry and fisheries, earth observation and remote sensing, and communications and broadcasting. In particular, UAVs can accomplish missions that could be dangerous for humans,

(a) UAV equipped with 2 cameras and a LIDAR

(b) Picture of the third floor

Figure 1.1: Inspection of the unit 3 reactor of the Fukushima nuclear plant. Photos taken by Tokyo Electric Power Company Holdings, Inc on 27 February 2018

such as inspection of remote places or search and rescue operations. Domains of application include inspection of nuclear plants [9], mapping of uranium mines [10], mapping of contaminated zones [11] or bridge inspection [12]. For instance, a UAV was used in 2018 to check the radioactivity inside the Unit 3 of the damaged nuclear plant at Fukushima Daiichi[1]. The UAV along with one of the taken picture from it are shown in Figure 1.1. The mission of the UAV was to estimate the radioactivity level at three different floors of the reactor. This example illustrates an environment that is not only difficult to access, but also dangerous for a human. The flight capabilities of the drone allowed the UAV to cover different floors without risking to interact with the building. The UAVs market shares for the military, commercial and consumer applications were 70 %, 17 % and 13 % respectively [13] in 2018. While the military is still the main domain of application, the non military UAV production is projected to grow from $2.6 billion in 2016 to $10.9 billion in 2025 [14] highlighting the importance of tackling the challenges related to UAV operations. This thesis is focused on some of those technical challenges.

UAV missions can be classified into two categories: fully autonomous, without any human intervention, and teleoperation, where the UAV is manually flown by a human pilot using a remote controller. Certain tasks, such as bridge inspection to detect cracks, are traditionally carried out by a human operator, and UAVs can be used to make this process

[1] `http://www.tepco.co.jp/en/nu/fukushima-np/handouts/2018/images/handouts_180301_01-e.pdf`

safer, faster, and cheaper [12]. In this thesis, the problem of teleoperation, which still requires a human is addressed. The reason is that, while there is work towards fully autonomous missions (e.g., for automatic detection of cracks on a bridge), keeping a human operator to analyse the data and make ultimate decisions requires a minimal modification of the existing teleoperation protocol. It is not clear if or when fully autonomous missions will become possible for all applications of UAVs. It is thus important to continue improving teleoperation capabilities [15].

UAVs are of two categories: rotary wings and fixed wings. The design of rotary wings UAVs allows them to be more flexible compared to their fixed wings counterpart (e.g., hovering at the same spot, easier take off and landing), which is why this thesis focuses on them. They, however, are slower as a consequence, and have a reduced operational range.

1.2 Problem statement

The main advantages of a rotary wings UAV compared to a manned aircraft is its agility, low cost, and safety for the operator. However, remote control means that the operator loses access to the onboard human senses (e.g., no acceleration feeling, limited field of view and resolution), and has to rely on the onboard sensors. The reduction in feedback for remotely controlled UAVs makes human errors more likely to occur, even exceeding the mechanical failures [16, 17]. While the focus of this thesis is on UAVs, the same problem exists for other kind of robots: the Center for Robot-Assisted Search and Rescue, which has sent rescue robots in several incidents such as the World Trade Centre in 2001, Hurricane Katrina in 2005, and the Japanese Tsunami in 2011, published a study in 2010 identifying barriers to the success of the missions that occurred, among other, in the World Trade Centre with ground robots. Two challenges were identified: the difficulty to navigate the ground robot, and the difficulty to interpret the output of the sensors [18]. While the interpretation of the sensors data remains an issue for flying robots, the naviga-

tion problem is even more complicated since UAVs are naturally unstable, unlike ground robots. Thus, increasing the safety of UAV teleoperation can be done by improving the interface between the UAV and the operator through a flight assistance system, so that sensory data is easier to interpret, and the UAV is easier to manoeuvre.

It is however critical to ensure that this flight assistance system gives reasonable control to the operator during the flight. Two recent aircraft crashes of the Boeing 737 Max due to an anti-stall system highlight the importance of an interface that is both non invasive and easy to disable. The later means that the UAV should be able to fly normally without the system. Even though it is outside of the scope of this thesis, a second concern to be aware of for a flight assistance system is the influence of the system on the operator's skill. For manned aircraft, the increased use of automation raises questions on the capacity of pilots to operate without the automation [19, 20].

The physical interface between the operator and the UAV adopted in this thesis is a three dimensional haptic controller. Such a device allows to use haptic cues, which provide guidance or information back to the operator. Moreover, it is hypothesised that this device might be easier to use than a standard radio controller: the simplicity of displacing the end effector to move the drone compares favourably to separated control of thrust, roll, pitch and yaw when using a typical radio controller. This choice is a step in the direction of intelligent human-machine interface that is easier to use.

1.2.1 Aim

In this thesis, the overall aim is to increase the safety and efficiency of UAV teleoperation. In this context, the safety of a UAV means eliminating or reducing the possible collisions between the UAV and its environment. Such collisions can lead to human and material damages. The risk of collision increases when the environment is cluttered. Thus, teleoperation of UAVs in this kind of environment is considered as the most challenging case, hence this case is the one dealt with in this thesis. It is proposed to achieve this objective by assisting the operator with a flight assistance framework whose main role is obstacle

avoidance. Moreover, an intelligent flight assistance system might broaden the use of UAVs by allowing less trained operators to fly safely.

The main objective is to provide a complete system architecture for obstacle avoidance, including registration of human inputs, obstacle detection, obstacle representation and appropriate reaction of the UAV to dangerous obstacles. The sensors considered for this task are LIDAR sensors and/or stereo cameras. The interface between the operator and the UAV is a 3D haptic controller.

It is important to note that, due to the context of indoor teleoperation chosen for this thesis, the proposed algorithms are designed for UAVs flying at low velocity and might not be suitable for fast UAVs. The final algorithm of this thesis, OAST, should allow velocities of approximately $3\,\mathrm{m\,s^{-1}}$ and $10\,\mathrm{m\,s^{-1}}$ for sensing range of $2\,\mathrm{m}$ and $8\,\mathrm{m}$ respectively. More details on this topic can be found in section 5.8.1. Moreover, this thesis does not formally address issues related to the stability of teleoperation, which are discussed in section 5.8.2. In particular, stability is less of a concern for the OAST algorithm.

1.3 Thesis outline

The first area of investigation is related to the exclusive use of the haptic feedback provided by the haptic device to perform obstacle avoidance, which is detailed in **Chapter 2**. This chapter presents an algorithm to convert a point cloud directly into a haptic feedback whose goal is to repel the UAV away from obstacles. A simulated experiment is carried out with a realistic simulator and an ideal LIDAR sensor to validate the proposed method with human participants. Results show that this method improves the safety of teleoperation and can reduce the workload of the operator under certain tuning conditions. In order to further reduce the number of collisions of the UAV and improve manoeuvrability, it is hypothesised that a flight assistance system based on filtering the operator's inputs might perform better. Such a system is ultimately presented in **Chapter 5**.

In order to develop this filtering algorithm, a precise map of the environment is re-

quired. Indeed, such an approach relies on the location of the obstacles to act directly on the trajectory of the UAV. It is thus important to estimate precisely the location and shape of the surrounding obstacles. **Chapter 3** investigates the problem of mapping the area around the UAV to obtain a mobile map. The Normal Distribution Transform Occupancy Map [21] makes the mapping task possible. However, the obtained map is immobile, restraining the operational range of the UAV. This method is extended in this thesis to be UAV centric, allowing unlimited movement of the UAV. The quality of the map is tested through an odometry metric on two publicly available datasets and the proposed method provides competitive results. The sensor used in this chapter is a LIDAR sensor, which provides a precise but sparse point cloud.

Given the context of obstacle avoidance, increasing the density of the output of the sensor used would allow improving the detection capabilities of the UAV. Thus in **Chapter 4**, a fusion scheme between LIDAR and stereo camera data is proposed. The objective is to get a denser point cloud by combining data from a stereo camera and a LIDAR sensor. This method is tested on a public dataset, and yields results well competitive with other fusion methods.

The map now allows to filter the operator's input to prevent collisions with obstacles and provide better manoeuvrability. The whole system is described in **Chapter 5**. Human experiments in a realistic synthetically simulated environment reveal that under perfect odometry, the safety is ensured: the collisions are all but eliminated. Moreover, the system also increases efficiency while reducing workload for the operators, achieving the objective of a non invasive flight assistance system.

Finally, **Chapter 6** draws conclusions from this research and proposes possible improvements in future works.

1.4 Contributions

A summary of the contributions of this thesis is provided in this section.

1. In Chapter 2,

   (a) A new obstacle avoidance algorithm using haptic feedback computed from a point cloud in 3D is proposed.

   (b) This algorithm is evaluated with a realistic open source simulator in experiments involving the participation of 11 untrained people, using an ideal LIDAR. An analysis of the results is provided.

2. In Chapter 3,

   (a) NDT RC, a new algorithm allowing the movement of a map with the robot is proposed, extending an existing mapping algorithm in which the map was not mobile. The raytracing part of the algorithm is also improved.

   (b) NDT RC is evaluated on two publicly available datasets using an odometry metric.

3. In Chapter 4,

   (a) A new algorithm to fuse LIDAR and stereo data in order to create denser depth maps is proposed.

   (b) An evaluation of the quality of the depth map from this algorithm is performed on a publicly available dataset. Moreover, a comparison against several other algorithms which are applied to this dataset is proposed. This comparison provides baseline results between algorithms to fuse stereo and LIDAR data.

   (c) The influence of the addition of the depth map to the mapping algorithm is studied with a particular focus on the density and precision of the map. The impact on runtime is provided as well.

4. In Chapter 5,

   (a) A new obstacle avoidance algorithm based on the filtering of the operator's input is developed. This algorithm also provides trajectory assistance to the

operator, which is demonstrated through simulations.

(b) OAST, a complete software system running in real time incorporating the map created in Chapter 3 is developed. This system receives commands from the operator through the 3D haptic controller and LIDAR data. The output is a filtered input from the operator and an indicative haptic feedback.

(c) This flight assistance system is tested with a realistic open source simulator in an experiment involving the participation of 20 people, assuming a perfect odometry. An analysis of the results is provided.

This thesis contributed to several publications: Chapter 2 led to the work presented in Courtois and Aouf [1], Chapter 4 was the basis of the work Courtois and Aouf [2]. The content of Chapter 3 and Chapter 5 has been submitted for publications [3, 4].

1.5 Publications resulting from this work

[1] Hugo Courtois and Nabil Aouf. “Haptic feedback for obstacle avoidance applied to unmanned aerial vehicles”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 417–424.

[2] Hugo Courtois and Nabil Aouf. “Fusion of stereo and Lidar data for dense depth map computation”. In: *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE. 2017, pp. 186–191.

[3] Hugo Courtois, Nabil Aouf, Kenan Ahiska, and Marco Cecotti. “NDT RC: Normal Distribution Transform Occupancy Map with recentering”. Submitted to IEEE International Conference on Robotics and Automation.

[4] Hugo Courtois, Nabil Aouf, Kenan Ahiska, and Marco Cecotti. “OAST: Obstacle Avoidance System for Teleoperation of UAVs”. Submitted to IEEE Transactions on Human-Machine Systems (THMS).

Chapter 2

Obstacle avoidance through haptic feedback

Contents

In this chapter, the use of a 3D haptic controller to manoeuvre the UAV while avoiding obstacles is examined. An algorithm to generate a haptic feedback based on the position of obstacles surrounding the UAV is proposed. This algorithm is tested with a group of volunteers using a 3D haptic controller in a realistic simulated environment. The results are statistically analysed. This experiment shows that this method can reduce the number of collisions. It is as well possible to decrease the operator's workload at a safety cost by using the proposed hole detection algorithm.

2.1 Introduction

The main idea of this chapter is to use the haptic feedback to help the operator localise and avoid the obstacles. The method proposed is to control the UAV with a 3D haptic controller. The haptic feedback is steering the end effector in a direction opposite to the obstacle. It is then possible for the operator to localise obstacles outside of the field of view, and if the obstacle is in the field of view, the feedback gives a second sense of distance to the obstacle.

2.1.1 Haptic controller

Haptic technology aims to recreate the sense of touch by sending mechanical feedback to the user (e.g., force or vibration). It is used in various domains such a computer-aided design, telesurgery, or video games [22].

Figure 2.1: Pictures of 3D haptic controllers. From left to right: Novint Falcon, Geomagic Touch, Omega.3

This feedback can be used to transmit information to the user, thus a haptic controller can be seen as a bi-directional sensor, which both receives and sends information.

Since a quadrotor has 4 controllable degrees of freedom, haptic controllers with at least 3 degrees of freedom are considered. The last degree of freedom (usually yaw) can be controlled either by buttons or a stylus attached to the effector depending on the controller model. The two major kinematic structures are illustrated in Figure 2.1: the Novint Flacon and Omega.3 exhibit a delta robot configuration while the Geomagic touch features a serial design.

2.1.2 Objective and prior work

The objective of this chapter is to improve the situational awareness of the operator when piloting small UAVs. Two assumptions are made in this chapter :

(i) the UAV is control augmented, which means that a controller stabilises the attitude dynamics of the UAV, leaving to the operator the control of the position through the

controller,

(ii) the obstacles are detected in 3D.

The first assumption is reasonable since there is now a wide range of position controller available for UAVs (e.g., [23, 24]). The second assumption is more challenging, but realistic in a multi modalities setup, as seen in Chapter 4 or with systems equipped with more sensors [25].

There is literature about the use of haptic feedback through the controller of the robot for obstacle avoidance. This method is successfully applied to ground robots [26], and the concept has been extended to UAVs (e.g., [27–29]).

In order to perform obstacle avoidance through haptic feedback for autonomous ground robots, artificial force fields have been used (e.g., the distance between the obstacle and a robot arm is used to create the field [30]). These artificial fields for UAVs have been studied [27], notably the Parametric Risk Field (PRF). A comparison mixing force and stiffness feedback using these force fields took place using 2D joysticks [31]. A force feedback applies a force directly to the end effector, so that the end effector is actively deflected. On the other hand, a stiffness feedback modifies the spring constant that links the end effector to the centre of the controller workspace. The work of Lam et al. [31] concludes that haptic feedback is able to improve safety and with proper tuning, can also reduce workload for the operator. Even though this work took place in three dimensions, the obstacle detection and haptic feedback was limited to two dimensions only.

A comparison of four force feedback algorithms[29] has found the most efficient to be the “Time To Impact” (TTI). The TTI algorithm computes the time before an impact by considering the distance between the UAV and the obstacle and the current velocity of the UAV. The haptic force is then inversely proportional to the the time to impact. In a following study, [32], a pure stiffness algorithm is introduced. The authors show that this algorithm both reduces the number of collisions and the workload of the operator. A limitation of this method is that the obstacles are only detected in 6 directions, which means that obstacles not in the x, y, or z directions do not generate any feedback. More-

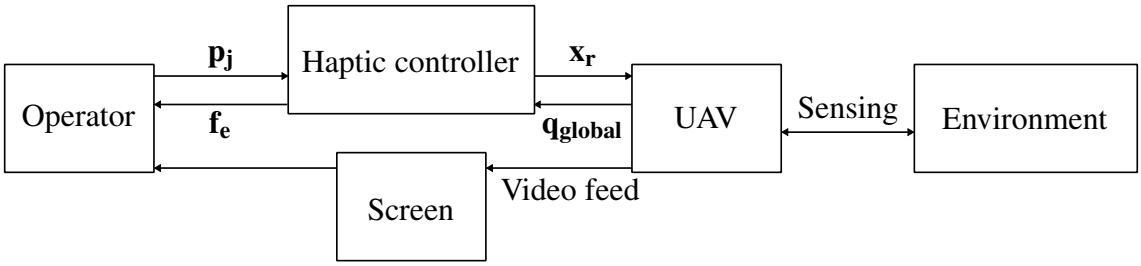


Figure 2.2: Overall description of the haptic scheme. The displacement of the end effector of the haptic controller is $\mathbf{p_j}$, the desired position goal of the UAV is $\mathbf{x_r}$, the avoidance vector computed from the position of the surrounding obstacles is $\mathbf{q_{global}}$ and the force vector generated by the haptic controller is $\mathbf{f_e}$

over, this is a pure stiffness feedback, thus if the displacement of the controller is small, the feedback will also be small. A pure stiffness approach is also vulnerable to moving obstacles for this same reason: if the controller is not deflected, and a moving obstacle gets close, then the user does not feel any feedback.

The methods presented above are not tied to a particular detection system and assume that obstacle detection has already been done. Some methods linking obstacle detection and haptic feedback exist (e.g., the haptic force can be directly derived from the optical flow through the concept of optical impedance [33]).

The aim of this work is to design an obstacle avoidance system which meets the following functional criteria:

(i) is able to detect the obstacle in all directions of the space (complete 3D),

(ii) is able to avoid obstacles in absence of user input, while stabilising in a safe place,

(iii) allows flight into small closed spaces.

The overall method is described in Figure 2.2. The operator acts on the haptic controller by moving the end effector. The position of this effector is then used to set a position goal for the UAV. The UAV uses its onboard sensors (in this chapter, a LIDAR sensor is considered, in addition to the camera providing the video feed) to probe the environment and computes an avoidance vector $\mathbf{q_{global}}$. This vector is converted into a force feedback sent to the operator. The operator also has access to a video feed to perceive the

surrounding of the UAV.

Section 2.2 presents the artificial force field (to compute $\mathbf{q_{global}}$) and the method used to transform the output of the force field into a haptic feedback for the controller (i.e. convert $\mathbf{q_{global}}$ into $\mathbf{f_e}$). Section 2.3 explains how the experiment was conducted to test the proposed design, then Section 2.4 shows the results of this experiment.

2.2 From artificial force field to haptic feedback

2.2.1 Mapping from the controller to the UAV

The mapping from the controller displacement to the UAV position is explained in this section. The challenge is to map the displacement of the controller, which is finite, to the displacement of the UAV, that is unconstrained. In this context, the movement is said to be constrained if there exists a distance L such that, for all positions $\mathbf{x_r}$ of the UAV:

$$\|\mathbf{x_r}\| < L. \tag{2.1}$$

A solution to this issue is to map the velocity or displacement of the controller to the roll, pitch, and thrust of the UAV [34]. The issue with this solution is that it is difficult to perform small movements, since the end effector has to be precisely at the centre of the workspace for the UAV to stand still. A technique based on sliding mode is proposed in [35], however, a button needs to be pressed to switch between the position mode and the sliding mode. In [36], a scheme is proposed which allows both small precise displacement and big displacement, without the need of a manual switch. This is the scheme that is adopted in the proposed setup. A brief description is included in this chapter for completeness : let $\mathbf{p_j}$ be the controller displacement, r^* be a radius, $\mathbf{K_p}$ and $\mathbf{K_v}$ be diagonal positive definite scaling matrices and $\mathbf{x_r}$ the reference position of the UAV in real space,

then the mapping from $\mathbf{p_j}$ to $\mathbf{x_r}$ is as follows:

$$\mathbf{x_c}(t) = \int_0^t \mathbf{K_v}\mathbf{p_j}(s) \max\left(0, 1 - \frac{r^*}{\|\mathbf{p_j}(s)\|}\right) \mathrm{ds}, \tag{2.2}$$

$$\mathbf{x_r}(t) = \mathbf{K_p}\mathbf{p_j}(t) \min\left(1, \frac{r^*}{\|\mathbf{p_j}(t)\|}\right) + \mathbf{x_c}(t). \tag{2.3}$$

The principle is to define a virtual sphere of radius r^* centred on the centre of the controller workspace. $\mathbf{x_c}$ is the centre of the virtual sphere, and $\mathbf{x_r}$ is the position of the UAV: when the end effector of the controller is inside this sphere ($\|\mathbf{p_j}\| < r^*$), the UAV moves around $\mathbf{x_c}$, which is constant. When the end effector goes outside of the sphere, $\mathbf{x_c}$ starts moving, thus allowing unconstrained movement for the UAV. This scheme can be seen as a proportional mapping when the end effector is not deflected much ($\|\mathbf{p_j}\| < r^*$) and an integral mapping otherwise. Having the precise control when the end effector is near the middle of the workspace is interesting because some haptic devices exhibit a higher resolution near the centre of the workspace (such as the Novint Falcon [37]), in addition to the main requirement of unlimited movement.

2.2.2 The Parametric Force Field

The first stage of the avoidance process is to map the presence of obstacles to a repulsion vector, one vector per obstacle. To this end, an artificial force field is used. A good candidate is described in the work of Lam et al. [27] under the name "Parametric Risk Field" (PRF), that is adopted in this work. The PRF is interesting because it deals with two issues of potential fields applied to UAVs:

- the potential at the origin becomes infinite, which is not possible on a real controller,
- the velocity of the UAV is not taken into account, only the distance (i.e. a repulsive force can be generated by moving away from an obstacle).

The PRF solves the infinite potential issue by adding a critical zone, so that there is a

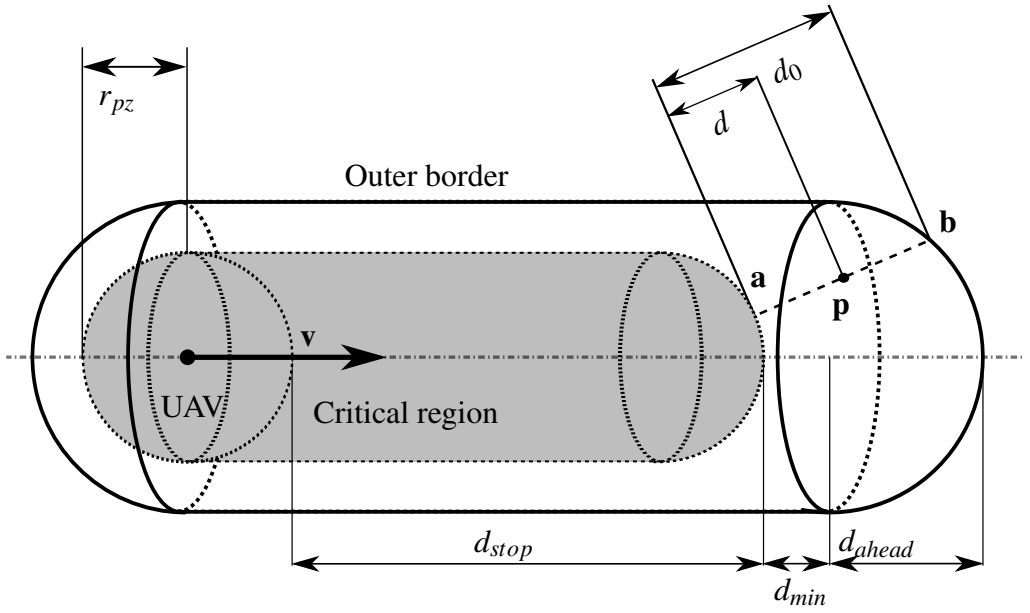


Figure 2.3: 3D shape of the PRF for a given velocity

maximal threshold for the feedback force. In order to take the velocity into account, the force field is oriented in the same direction as the velocity vector and its dimension along this direction depends on the velocity. Moreover, the constant width of the PRF is an advantage in closed space scenarios (e.g., in corridors), because only obstacles in the direct trajectory of the UAV or near this trajectory are considered to compute the feedback, which reduces computational load and diminishes the risk of false positive. In order to extend the PRF to 3D (Figure 2.3), the borders are treated as cylinders with rounded ends.

The value returned by an obstacle in the PRF represents the risk of collision with this obstacle. Let $\mathbf{p}$ be the position of an obstacle, $\mathbf{v}$ be the velocity of the UAV and $f : [0,1] \to [0,1]$ be a continuous decreasing function such that $f(0) = 1$ and $f(1) = 0$. Then the value PRF returned by the force field is (see Figure 2.3 for a representation of the different zones of the PRF):

$$\mathrm{PRF}_{\mathbf{v}}(\mathbf{p}) = \begin{cases} 0 & \text{if } \mathbf{p} \text{ is outside of outer border,} \\ 1 & \text{if } \mathbf{p} \text{ is inside critical region,} \\ f(\frac{d}{d_0}) & \text{otherwise.} \end{cases} \tag{2.4}$$

The lengths d and d_0 are illustrated in Figure 2.3 for a particular obstacle $\mathbf{p}$. Their ratio indicates where the obstacle is in between the critical region ($d/d_0 = 0$) and the outer border ($d/d_0 = 1$). Let $D_\mathbf{p}$ be the line orthogonal to the outer surface which passes through the obstacle $\mathbf{p}$. Let $\mathbf{a}$ be the intersection between $D_\mathbf{p}$ and the outer surface of the critical region, and $\mathbf{b}$ be the intersection between the outer border and $D_\mathbf{p}$. Then $d = \|\mathbf{a}-\mathbf{p}\|$ and $d_0 = \|\mathbf{b}-\mathbf{a}\|$.

The constants used in the computation of the PRF are as follows: r_{pz} represents half the width of the critical region, d_{min} represents the minimal distance between the critical region and the outer border, a_{max} represents the maximum deceleration of the UAV and t_{ahead} represents the time that the operator has to react to an obstacle. From those four constants, the parameters of the PRF are computed as follows:

$$d_{stop} = \frac{\|\mathbf{v}\|^2}{2a_{max}}, \tag{2.5}$$

$$d_{ahead} = \|\mathbf{v}\| t_{ahead}. \tag{2.6}$$

In practical terms, d_{stop} represents the distance needed to stop the UAV whilst applying maximal deceleration and d_{ahead} represents the distance that the UAV can cross at its current speed during t_{ahead}. The rate of change of the PRF is given by the f_P function. In [27], it was noted that the derivative of a shifted cosine function is continuous at $d/d_0 = 1$ while being almost linear from $d/d_0 = 0.4$. This makes such a function more suitable than a simple linear function. Thus the function f_P that is used in the PRF is:

$$f_P\Big(\frac{d}{d_0}\Big) = 1 + \cos\Big(\frac{d}{d_0}\frac{\pi}{2} + \frac{\pi}{2}\Big). \tag{2.7}$$

For a graphical representation of f_P, see Figure 2.4.

The decaying rate of the PRF can be seen in Figure 2.5 for different speeds, with parameters $r_{pz} = 1\,\mathrm{m}$, $d_{min} = 0.8\,\mathrm{m}$, $t_{ahead} = 1.5\,\mathrm{s}$ and $a_{max} = 0.5\,\mathrm{m\,s^{-2}}$.

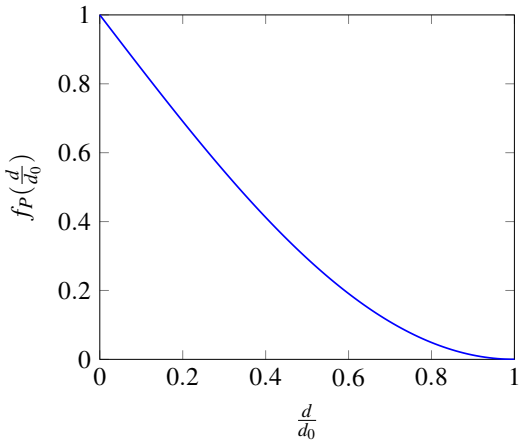


Figure 2.4: Plot of the f_P function

The detailed implementation of the PRF in 3D is described in Algorithm 2.2, assuming without loss of generality that the UAV is located at the origin and that its velocity is aligned with the x axis. The function used to check if an obstacle is inside the PRF in Algorithm 2.2 is detailed in Algorithm 2.1.

Algorithm 2.1: Algorithm to check if a point is located inside a PRF shaped field: *isInsideField*

Input : the position of an obstacle $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}$,
radius of the field r,
length of the field l.

Output: A boolean that indicates if the point is inside the field.

Function `isInsideField(`**p**, r, l`)`**:**

`// First, check if the point is inside a cylinder a radius` r `around the` x `axis.`

if $p_y^2 + p_z^2 > r^2$ **then return** False;

`// Second, check if the point is inside the cylinder without the spheres.`

if $p_x \in [0, l]$ **then return** True;

`// Finally, check if the point is inside any of the two spheres at each end of the cylinder.`

$\mathbf{c_1} = \begin{bmatrix} l & 0 & 0 \end{bmatrix}$;

if $\|\mathbf{c_1} - \mathbf{p}\|^2 \leq r^2$ ***or*** $\|\mathbf{p}\|^2 \leq r^2$ **then**

  return True

end

return False;

Algorithm 2.2: Algorithm to compute the intensity of a point inside the PRF

Input : the position of an obstacle $\mathbf{p} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}$,
the half width of the critical region r_{pz},
the minimal distance between the critical region and the outer border d_{min},
the minimal distance to stop d_{stop},
the look ahead distance d_{ahead}.

Output: the intensity $i_{\mathbf{p}}$ of the PRF associated to this obstacle.

$d_{tot} = r_{pz} + d_{min}$;
`// Check if the point is outside the PRF or inside the critical zone.`
if $\neg$`isInsideField(`$\mathbf{p}, d_{tot}, d_{stop} + d_{ahead}$`)` **then return** 0;
if `isInsideField(`$\mathbf{p}, r_{pz}, d_{stop}$`)` **then return** 1;
`// 3 cases: the point is in front of, between, or behind the critical zone`
if $p_x > d_{stop}$ **then**
  $\mathbf{l_o} = \begin{bmatrix} p_x - d_{stop} & p_y & p_z \end{bmatrix}$;
  $d = \max(\|\mathbf{l_o}\| - r_{pz}, 0)$;
  if $d = 0$ **then return** 1;
  $y^* = \sqrt{p_y^2 + p_z^2}$;
  $\theta = |\operatorname{atan2}(y^*, p_x - d_{stop})|$;
  $\theta_c = \operatorname{atan2}(d_{tot}, d_{ahead})$;
  if θ *is too small* **then**
    $d_0 = d_{tot} + d_{ahead}$
  else if $\theta \geq \theta_c$ **then**
    $d_0 = d_{tot} / \sin(\theta)$;
  else
    $d_0 = \sqrt{d_{tot}^2 - \sin(\theta)^2 d_{ahead}^2} + \cos(\theta) d_{ahead}$;
  end
  $d_0 = d_0 - r_{pz}$;
  $d_0 = \max(d_{min}, \min(d_{min} + d_{ahead}, d_0))$;
else if $p_x \in [0, d_{stop}]$ **then**
  $d = \sqrt{p_y^2 + p_z^2} - r_{pz}$;
  $d_0 = d_{min}$;
else
  $d = \|\mathbf{p}\| - r_{pz}$;
  $d_0 = d_{min}$;
end
return $f_P(d/d_0)$;

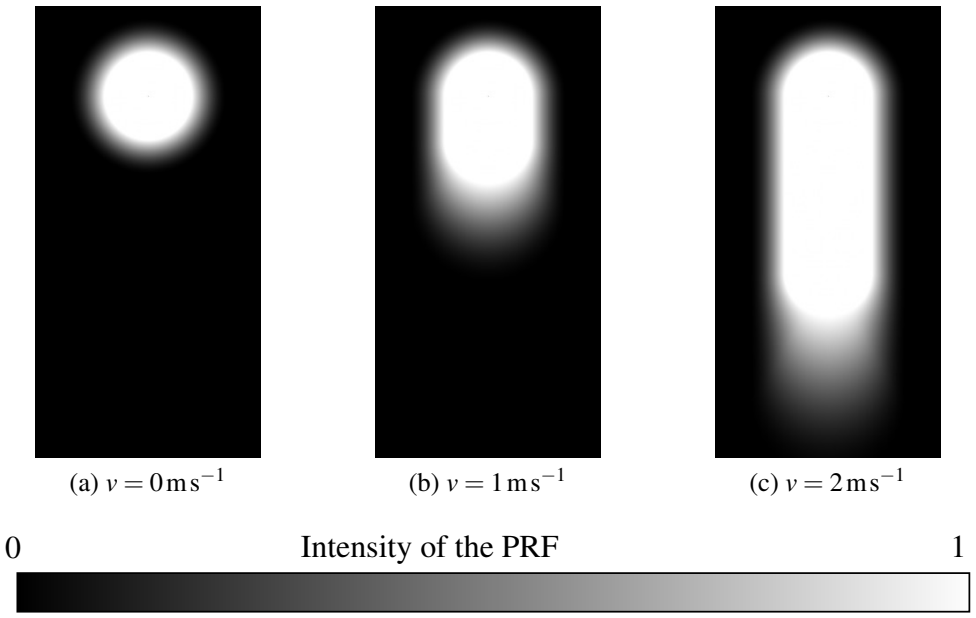


Figure 2.5: PRF shape for different velocities (vertical direction)

2.2.3 Mapping from the PRF to the avoidance vector

The PRF provides a risk value ranging from 0 (there is no obstacle in the field) to 1 (there is an obstacle in the field, close to the UAV) for each obstacle. The next goal is to fuse all those risk values into a single avoidance vector $\mathbf{q_{global}}$. In the following, an obstacle is defined as a 3D point in space (real objects such as walls or doors can thus be described as a combination of several obstacles).

First, for each obstacle, an obstacle vector whose origin is on the obstacle is computed, which points toward the UAV, and whose norm is the risk value from the PRF. Let O be the subset of $\mathbb{R}^3$ which contains those obstacle vectors. The objective is to combine the vectors in O into a single vector $\mathbf{q_{fused}}$. In order to avoid obstacles, it is desirable to stay away from the obstacles that generate the obstacle vectors with the highest norm. Among those dangerous obstacles, a suitable direction of avoidance has to be found. For instance, if two obstacles are diametrically opposed and generate an obstacle vector with the same norm, there is no need to generate a force in the direction defined by the two obstacles. On the other hand, if one of the obstacle is more dangerous than the other (i.e. it generates an obstacle vector with a higher norm), then it is desirable to generate a repelling force. More

precisely, the max function over a subset $P = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ of $\mathbb{R}^3$, where $\mathbf{x_i} = \begin{pmatrix} x_i, y_i, z_i \end{pmatrix}$ is defined as:

$$\max(P) = \begin{pmatrix} \max(x_1, \ldots, x_n, 0) \\ \max(y_1, \ldots, y_n, 0) \\ \max(z_1, \ldots, z_n, 0) \end{pmatrix}. \tag{2.8}$$

The min function is defined in a similar way. All the obstacle vectors are fused into $\mathbf{q_{fused}}$ using the technique presented in [27], which will be called *min/max* in the following:

$$\mathbf{q_{fused}} = \min(O) + \max(O). \tag{2.9}$$

In other words, all the obstacle vectors in O are fused into $\mathbf{q_{fused}}$ by keeping the extrema of the entire set of obstacle vectors in O. If the norm of $\mathbf{q_{fused}}$ is larger than 1, it is normalised. The *min/max* method has the advantage of keeping track of the biggest obstacle vectors. However, this method has a drawback: if there are symmetric obstacles with a hole in the middle (e.g., window, narrow corridor), the hole is not detected. This is illustrated in Figure 2.6: $\mathbf{o_1}$ and $\mathbf{o_2}$ are the biggest *obstacle vectors* detected by the PRF (the other obstacle vectors are not drawn for clarity reasons, there would be one vector per point belonging to the walls, but by definition, $\mathbf{o_1}$ and $\mathbf{o_2}$ generate the obstacle vectors with the highest magnitude). When equation (2.9) is applied, the x components of the obstacle vectors will cancel each other (they will not cancel completely if the UAV is not exactly in the middle, but the remaining x component would be small compared to the y component). The only remaining component is in the y direction. This is an issue because there is in fact no obstacle in this direction. This example with a corridor can be extended to any type of obstacle with a symmetric shape.

In order to correct this, a step is added to ensure that there are actual obstacles in the direction of $\mathbf{q_{fused}}$. The amount of obstacles which are both in the PRF and in the direction

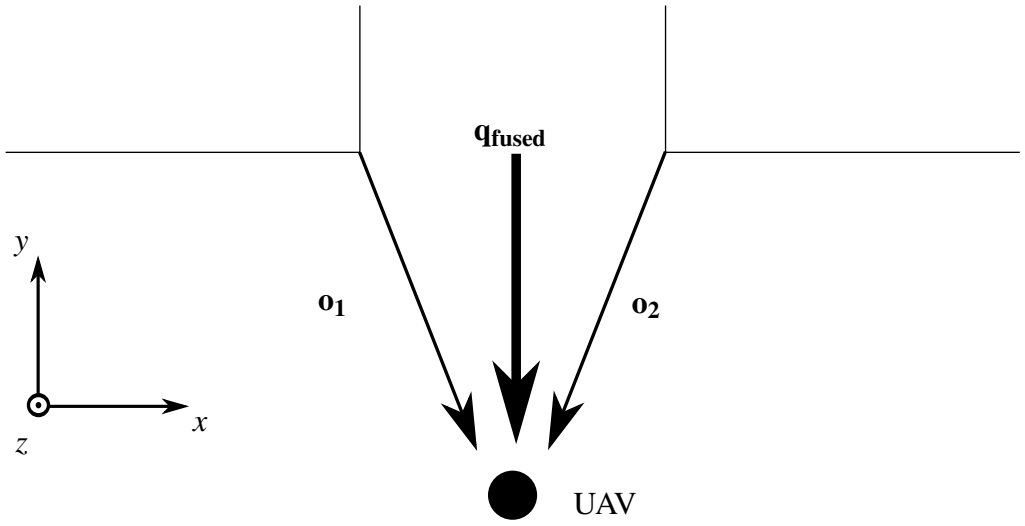


Figure 2.6: Illustration of an issue with the *min/max* method when encountering symmetric obstacles

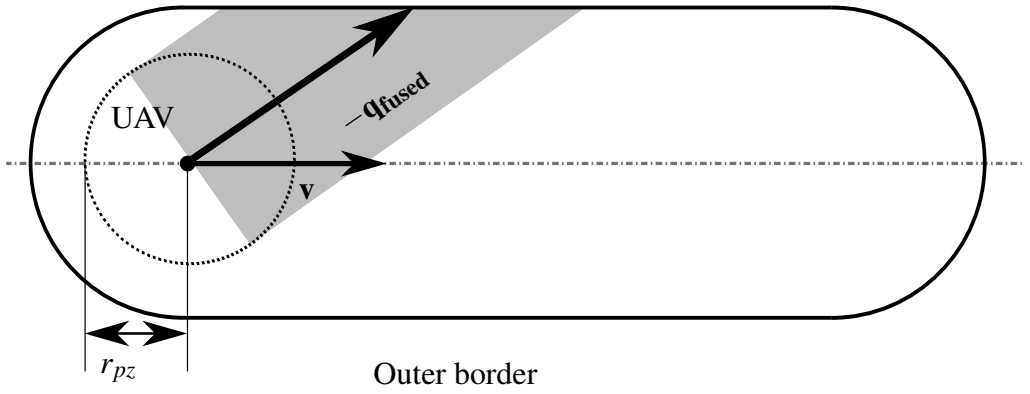


Figure 2.7: Zone inspected by the hole detection system

of $\mathbf{q_{fused}}$ is counted. In Figure 2.7, outer bound of the PRF and the $\mathbf{q_{fused}}$ direction are drawn. The number of obstacles in the grey area is computed, let it be called n_{obs}.

From here, a criterion is defined to determine whether there is a plausible obstacle in this zone. A threshold n_{lim} is empirically defined. The idea is that that when $n_{obs}/n_{lim} < 1$, the number of obstacles in the direction of $\mathbf{q_{fused}}$ does not support an actual obstacle in this direction. This method has a drawback: it is dependent on the resolution of the detection method, since the threshold has to be adapted to the resolution of the detecting sensor. In order to apply the threshold, $\mathbf{q_{fused}}$ is scaled by an increasing function $g(n_{obs}) : \mathbb{N} \rightarrow [0,1]$. There are several choices for g, the important points being continuity and quickly

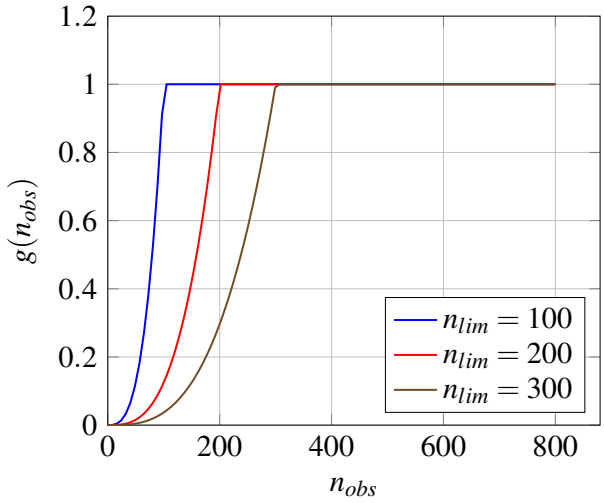


Figure 2.8: Scaling function g used in the hole detection system

decaying below n_{lim}. In this work, g is defined in equation (2.10).

$$g(n_{obs}) = \min\left(\left(\frac{n_{obs}}{n_{lim}}\right)^3, 1\right). \tag{2.10}$$

g is plotted in Figure 2.8 for different values of n_{lim}.

The $\mathbf{q_{fused}}$ vector is then scaled with the g function to obtain the *avoidance vector* $\mathbf{q_{global}}$:

$$\mathbf{q_{global}} = g(n_{obs})\mathbf{q_{fused}}. \tag{2.11}$$

The vector $\mathbf{q_{global}}$ can be interpreted as a safe direction, and its magnitude indicates how dangerous the obstacles around the UAV are.

2.2.4 From the avoidance vector to the haptic feedback

Once the avoidance vector $\mathbf{q_{global}}$ is computed, it is transmitted to the haptic controller. The haptic force applied to the end effector needs to reflect $\mathbf{q_{global}}$.

As noted in the work of Lam et al. [31], it is possible to combine force and stiffness feedback. The objective of the force feedback is to actively deflect the joystick from the

origin of the workspace while the stiffness feedback ensures that the haptic command is followed. Special care has to be taken so that the stiffness feedback does not hinder the force feedback [31]. This work was however done with a 2D joystick, by modulating the angle of deflection of the joystick. An algorithm for 3D haptic controllers is proposed in this section, that acts on the position of the end effector.

In order to encourage the user to follow the haptic feedback, a virtual spring is attached to the end effector. This spring pulls the end effector in a direction defined by the direction of the avoidance vector $\mathbf{q_{global}}$, with a force proportional to the norm of $\mathbf{q_{global}}$. In other words, this springs makes it more difficult for the user to travel against the direction defined by $\mathbf{q_{global}}$. Since active deflection of the end effector is a desirable feature, this spring is attached to the outer border of the workspace and not at the centre. In the following, this spring will be called the outer spring. A second virtual spring is added between the end effector and the centre of the workspace: the inner spring. The role of the inner spring is to give the user information about the location of the centre of the workspace, stabilise the end effector and bring back the end effector in the virtual sphere (see Section 2.2.1) so that in absence of human force, the UAV should end up stationary. For this reason, the spring constant of the inner spring is fixed. The resulting system is illustrated in Figure 2.9: the direction of the outer spring $\mathbf{e_q}$ is defined by the direction of $\mathbf{q_{global}}$. The outer spring intersects the workspace at point $\mathbf{c}$ and connects to the end effector $\mathbf{p}$. The spring constants of the inner spring and outer spring are k_i and $k_o = \alpha_o \|\mathbf{q_{global}}\|$ respectively. While k_i is constant, k_o depends on the magnitude of $\mathbf{q_{global}}$.

In order to visualise the forces acting on the joystick, Figure 2.10 shows the forces acting on the end effector when the the end effector is aligned with the direction of $\mathbf{q_{global}}$. In other words, this scenario represents the case where $\mathbf{p}$ is on the line between $\mathbf{c}$ and the centre of the workspace. The force on the main effector is illustrated for three different norms of the avoidance vector $\mathbf{q_{global}}$: 0, meaning that only the inner spring is active, 0.5 and 1.0, which is the maximum value for the norm of $\mathbf{q_{global}}$.

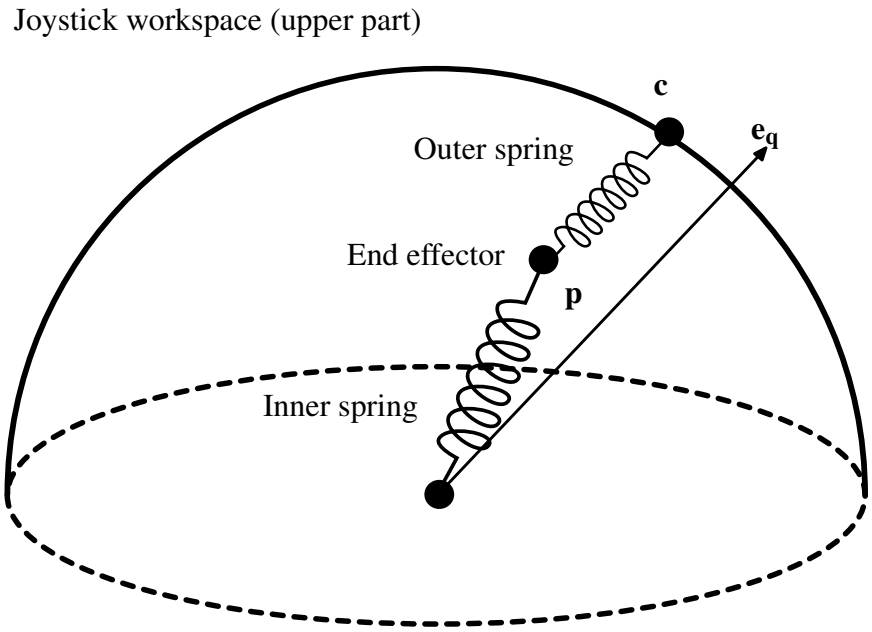


Figure 2.9: Schematic of the forces acting on the end effector inside the workspace

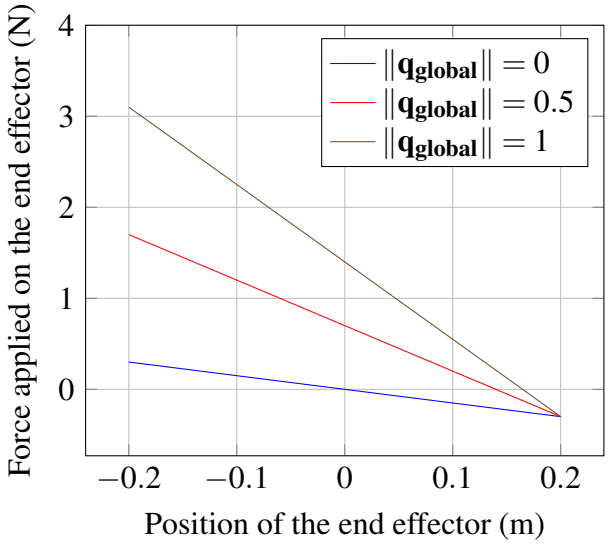


Figure 2.10: Forces acting on the end effector when it is aligned with $\mathbf{q_{global}}$. Spring parameters are $k_i = 1.5\,\mathrm{N\,m^{-1}}$ and $\alpha_o = 7\,\mathrm{N\,m^{-1}}$

There is an issue with such a system: when an obstacle is detected, the outer spring is there to actively deflect the controller in the opposite direction. However, if the operator wants to make an avoidance manoeuvre in other directions, the spring should not restrict this movement. In order to correct that behaviour, an additional term is added in k_o which represents the absolute value of the dot product between $\mathbf{e_q}$ and the displacement of the controller $\mathbf{p}$. The objective of this modification is to make displacements easier when they are not directly oriented in the direction opposite $\mathbf{q_{global}}$.

To summarise, let $\mathbf{q_{global}}$ be the avoidance vector obtained with equation (2.11) (written in the haptic controller workspace, a transformation into the reference frame of the UAV can be needed), k_i and α_o be user defined real positive constants, L be the radius of the controller workspace (considered to be spherical) and $\mathbf{p}$ be the displacement of the controller. Then the force $\mathbf{f_e}$ applied to the end effector is written in equation (2.12).

$$\mathbf{e_p} = \frac{\mathbf{p}}{\|\mathbf{p}\|} \tag{2.12a}$$

$$\mathbf{e_q} = \frac{\mathbf{q_{global}}}{\|\mathbf{q_{global}}\|} \tag{2.12b}$$

$$s = -\mathbf{p}\cdot\mathbf{e_q} + \sqrt{\|\mathbf{p}\cdot\mathbf{e_q}\|^2 + L^2 - \|\mathbf{p}\|^2} \tag{2.12c}$$

$$\mathbf{c} = \mathbf{p} + s\mathbf{e_q} \tag{2.12d}$$

$$k_o = \alpha_o \|\mathbf{q_{global}}\| |\mathbf{e_q}\cdot\mathbf{e_p}| \tag{2.12e}$$

$$\mathbf{f_e} = -k_i\|\mathbf{p}\|\mathbf{e_p} + k_o(\|\mathbf{c}-\mathbf{p}\|)\mathbf{e_q}. \tag{2.12f}$$

Such a scheme also allows for obstacle avoidance in absence of human input. Regarding parameters tuning, the outer spring should have a higher spring constant than the inner spring, thus it is recommended that α_o be greater than k_i.

This design fulfils our first two goals: the 3D PRF allows us to detect obstacles in all directions of space, while the force rendering in the controller is also omnidirectional. The third objective, allowing navigation in closed spaces, is tested in section 2.3.

2.2.5 Illustration of the haptic feedback created from different configurations of obstacles

In order to get a better visualisation, two examples are shown in Figure 2.11 and Figure 2.12. For those figures, the radius of the workspace of the haptic controller is set to 20 cm and the spring parameters are $k_i = 1.5\,\mathrm{N\,m^{-1}}$ and $\alpha_o = 7\,\mathrm{N\,m^{-1}}$. For clarity, the hole detection system is not used in those plots. In each figure, the upper plot represents the UAV with its velocity, in red. Several obstacles are shown, in black. The obstacles are represented as points to simulate an input comparable to LIDAR sensors. The PRF generates an individual avoidance vector for each obstacle as described in Section 2.2.2, plotted as a black arrow. Then those vectors are fused into $\mathbf{q_{global}}$ as explained in Section 2.2.3. The bottom plot illustrates the actual forces acting on the end effector of the haptic controller given $\mathbf{q_{global}}$. For several hypothetical positions of the end effector inside the workspace, the force applied to it are drawn in blue.

Figure 2.11 illustrates an obstacle that is wall shaped. On the top figure, the overall contribution of all the obstacles goes backward, which is expected. It is noted that since the UAV is roughly in the middle of the wall, there is no lateral contribution. Moreover, the obstacles outside the PRF do not contribute to $\mathbf{q_{global}}$. On the bottom figure, the potential forces are displayed in blue. If the user places the end effector in the direction of the wall, forward, a strong force oriented backward would act on the end effector. If on the contrary the operator operates the joystick in the half part of the workspace opposite the obstacle, then the force is less strong, facilitating the operation of the drone. The effect of the dot product introduced in equation (2.12e) is shown on the bottom plot: the red arrows are the force that would be applied to the end effector if this dot product was not added. The effect of this dot product is to reduce the force feedback when placing the end effector in directions that are different from the direction of $\mathbf{q_{global}}$. Since those directions are not supposed to be dangerous, it should be comfortable to use the end effector in those zones. The addition of this dot product reduces the force feedback in alternative directions from the obstacle, which makes it easier to navigate around obstacles.

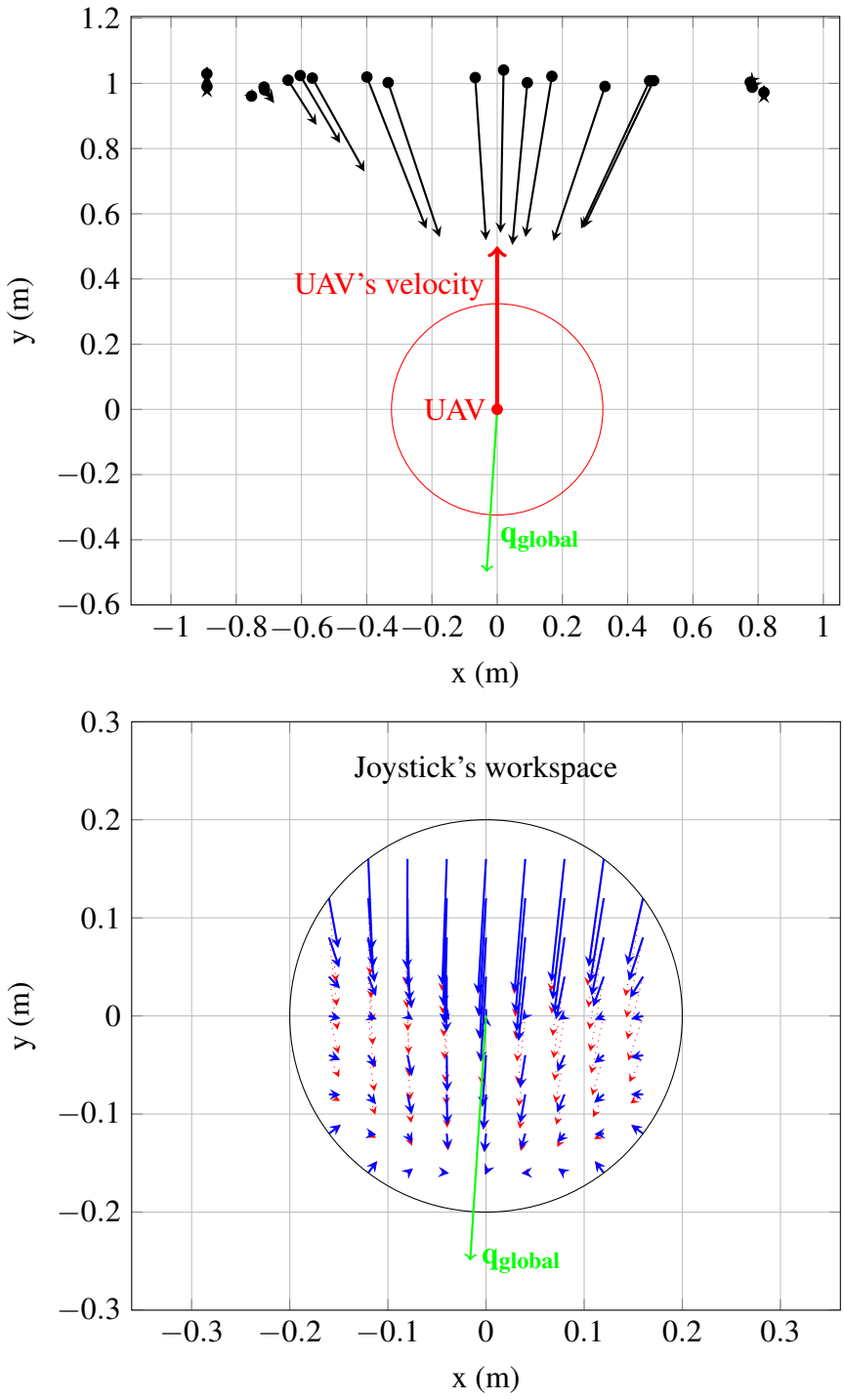


Figure 2.11: Representation of the forces applied to the end effector when facing a wall. Top: The UAV is represented in red, along with its velocity. The black points are obstacles, each of them generating an individual avoidance vector from the PRF, represented in black. After fusing those vectors, the green vector $\mathbf{q}_{\mathbf{global}}$ is obtained. Bottom: the workspace of the haptic controller. The vector $\mathbf{q}_{\mathbf{global}}$ is represented again for clarity. Each blue arrows represent the force that would be applied to the end effector if it was placed at the arrow's origin. The red arrows represent those same forces without the dot product in equation (2.12e)

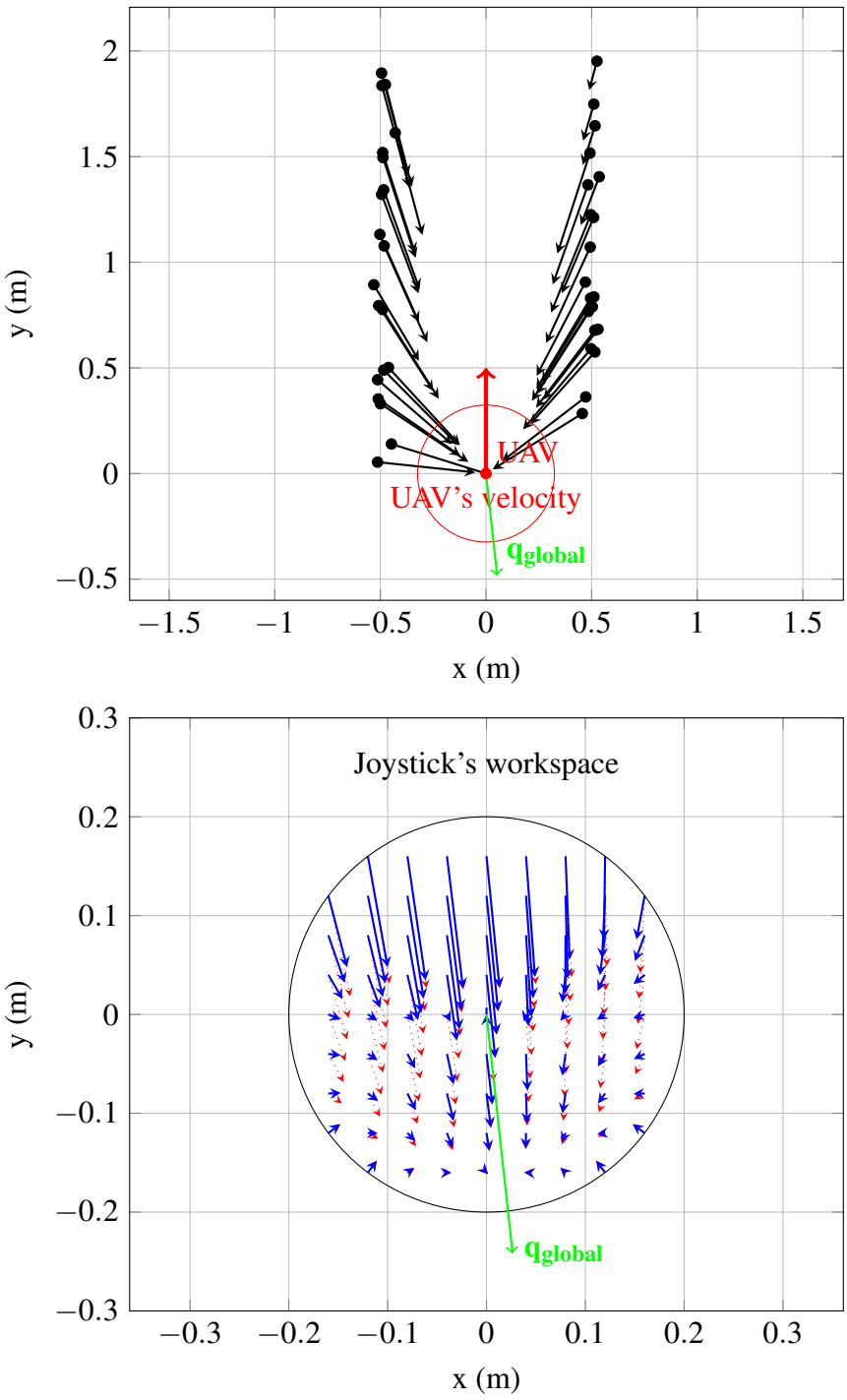


Figure 2.12: Representation of the forces applied to the end effector in a corridor. Top: The UAV is represented in red, along with its velocity. The black points are obstacles, each of them generating an individual avoidance vector from the PRF, represented in black. After fusing those vectors, the green vector $\mathbf{q}_{\mathbf{global}}$ is obtained. Bottom: the workspace of the haptic controller. The vector $\mathbf{q}_{\mathbf{global}}$ is represented again for clarity. Each blue arrows represent the force that would be applied to the end effector if it was placed at the arrow's origin. The red arrows represent those same forces without the dot product in equation (2.12e)

Figure 2.12 is a more detailed example of the phenomenon described in Figure 2.6. The UAV is placed at the beginning of a corridor, at equal distance of each walls. Due to the cancelled contribution along the y axis, the resulting avoidance vector $\mathbf{q_{global}}$ is oriented backward, which generates a backward repelling force in the joystick workspace. In this case, since no obstacle is situated in the direction of travel, the hole detection system described in Section 2.2.3 would cancel $\mathbf{q_{global}}$.

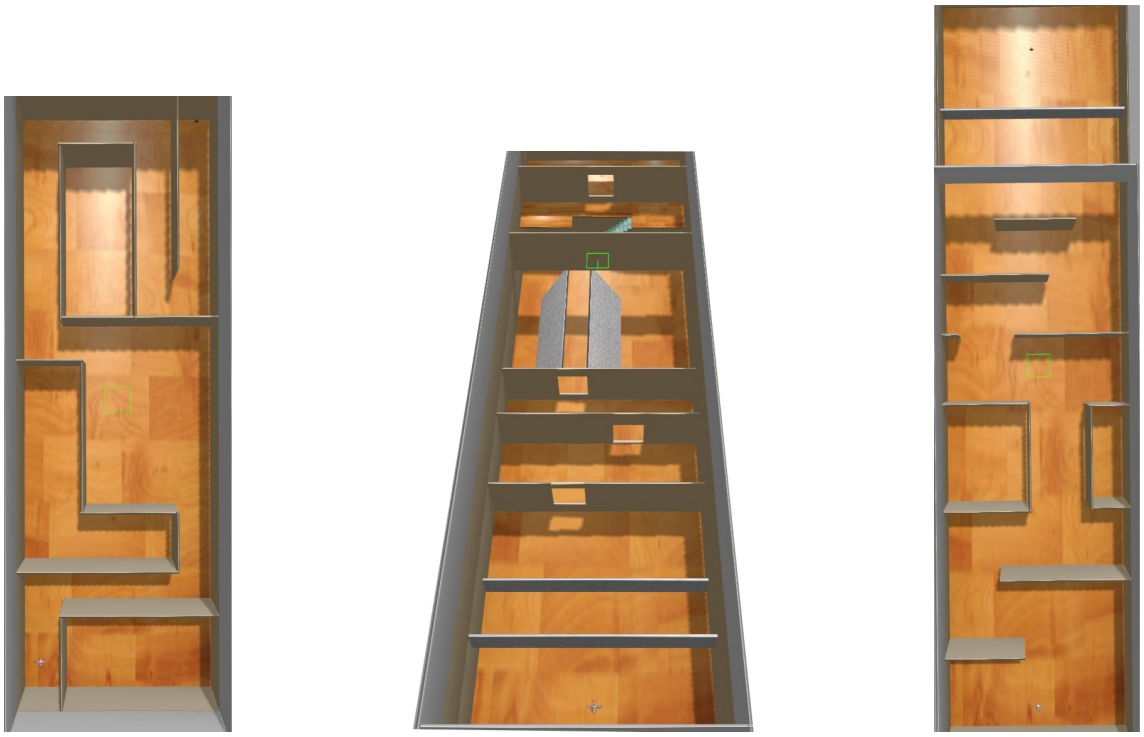
2.3 Experiment

2.3.1 Scenarios

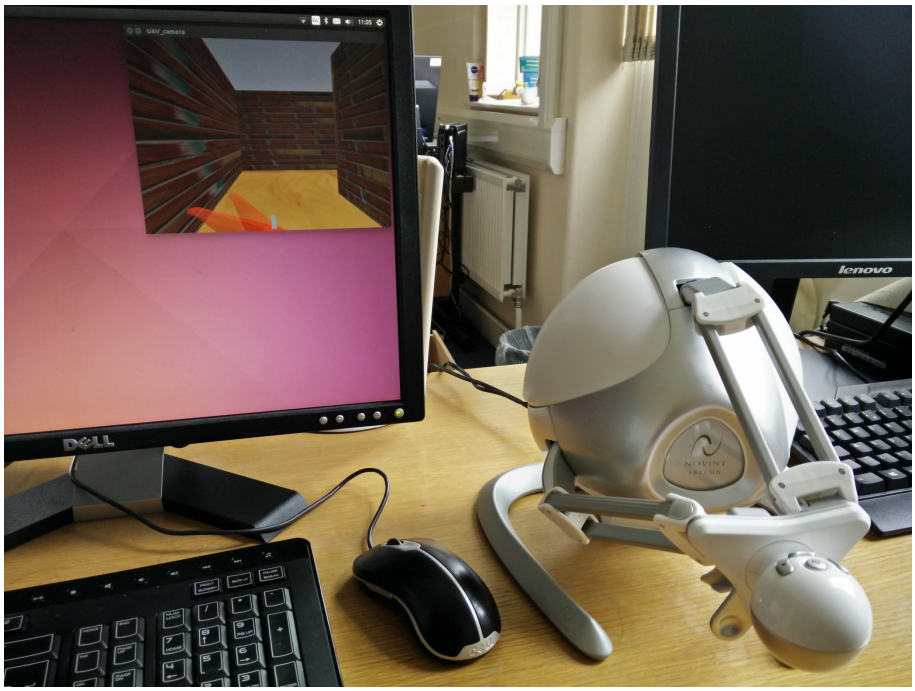
The method described in Section 2.2 is implemented in C++, using the Robot Operating System (Indigo) [38] middleware on Ubuntu 14.04. The simulator used is Gazebo [39], and the package which implements the UAV model is RotorS [40]. The haptic controller used is the Novint Falcon. Please note that this controller only has 3 degrees of freedom, thus the yaw is kept fixed. The library used to interface the haptic controller is HAPI[1] version 1.3 along with the *libnifalcon* drivers version 1.0.2[2]. The computer used in the experiment has an Intel Core i7 3930K processor, an NVIDIA GeForce GTX 680 graphic card and 16 GB of RAM.

In order to assess the efficiency of the method exposed, three scenarios have been created. In each of them, the goal is to fly the UAV from the beginning to the end. In all scenarios, the yaw angle of the UAV was fixed at 0. In the simulation, a camera is mounted on the UAV, with a resolution of 640×480 pixels and a field of view of 80°. In order to detect obstacles, a 3D LIDAR is implemented by combining 32 horizontal lasers, oriented between $-\pi/2$ and $\pi/2$, each laser having 256 points per scan. With this laser implementation, the method runs at 8 Hz. The total width of the UAV (including the propellers) is approximately 64.8 cm.

[1] `https://github.com/SenseGraphics/h3dapi/releases`
[2] `https://github.com/libnifalcon/libnifalcon`



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3



(d) Experimental setup

Figure 2.13: Top: Different scenarios used in the experiment (ceiling is not visible on those screenshots). Bottom: Experimental setup, the participants would only see the camera view. The haptic controller is visible on the right

Table 2.1: Values of the parameters used in the simulation

Parameter	**Value**	**Parameter**	**Value**
K_p	$\mathrm{Diag}(5,5,5)$	k_i	$1.5\,\mathrm{N\,m^{-1}}$
K_v	$\mathrm{Diag}(0.3,0.3,0.3)$	α_o	$7\,\mathrm{N\,m^{-1}}$
r^*	$0.025\,\mathrm{m}$	n_{lim}	200
a_{max}	$0.5\,\mathrm{m\,s^{-2}}$	r_{pz}	$0.7\,\mathrm{m}$
t_{ahead}	$2.0\,\mathrm{s}$	d_{min}	$0.3\,\mathrm{m}$

The scenarios are shown in Figure 2.13. The first focuses on manoeuvrability, with a succession of narrow corridors (2 m wide). The second aims to demonstrate the 3D capabilities of the method: the obstacles require the UAV to move in every direction of the space. The third scenario takes place in limited illumination (ambient light from 1 in scenarios 1 and 2 to 0.05 in scenario 3, where 1 corresponds to maximum luminosity and 0 to complete absence of light).

2.3.2 Experimental protocol

The values of parameters used in the algorithm are given in Table 2.1.

The experiment involved 11 people, none of which had previous experience using haptic joysticks. Each participant was asked to give informed consent before participating. During the briefing, each participant was asked to achieve the following two objectives for each scenario:

1. keep the number of collisions between the UAV and the environment as low as possible and

2. be as fast as possible.

Should a collision occur, the UAV would be frozen for 5 s, then brought back where it was 5 s before the collision. The different methods are:

1. without any feedback: m_0.

2. With the haptic feedback but without the g scaling function described in Section 2.2.3 (i.e. without the hole detection scheme): m_1.

3. With the complete haptic scheme: m_2.

The parameters recorded to assess the performance of the method are the number of collisions (N_{col}), the time to complete the run (T_c), the average velocity of the UAV (μ_{vel}) and the average distance to the nearest obstacle (D_{min}). The subjective workload is assessed using the NASA Task Load Index (TLX) [41] with pairwise comparison between the weights. A description of the NASA TLX evaluation can be found in annexe A.1.

Each participant would practice the three scenarios without any feedback (m_0) to get used to the controller. Those runs are not recorded. Then, each participant would go through the 3 scenarios with m_0, fill a NASA TLX form, and repeat the same operation with m_1 and m_2. The runs were grouped by method for two reasons: to minimise the learning effect of going through the same scenario back to back, and to allow participants to have a better appreciation of the method for the NASA TLX. This ordering was chosen because the objective is to evaluate the effect of adding haptic feedback: starting with m_1 or m_2 can affect the result of m_0 (e.g., after the feedback run, the user expects feedback but does not have it, therefore leading to a collision), but the opposite is less likely to happen, because m_0 does not modify the user experience (the control of the UAV stays the same). For this reason, the training run is only done using m_0, and does not use the feedback algorithms. This protocol yields 9 runs per person, for a total of 99 runs for the experiment (and 33 values for the NASA TLX evaluation).

Results were analysed with the R language [42], using generalised linear mixed models with the *lme4* [43] package. Each parameter was modelled separately using the scenarios and methods as fixed factors. The participants were modelled as random effects. Unless mentioned otherwise, a Gaussian distribution is used to create the models. The confidence level used for statistical significance is 0.95. The initial model was created by including an interaction term between method and scenario, which was dropped if its contribution to the model was not shown to be significant with the ANOVA. In case of

Table 2.2: Results of a full factorial ANOVA on the two fixed effects

Fixed effect	N_{col}	D_{min}	T_c	μ_{vel}	TLX
Method	$***$	$**$	$\cdot$	$\cdot$	$*$
Scenario	$***$	$***$	$***$	$***$	*NA*
Interaction method $\times$ scenario	$**$	$\cdot$	$\cdot$	$\cdot$	*NA*

p-values are indicated by '$***$', '$**$', '$*$' and '$\cdot$' if respectively $p \leq 0.001$, $0.001 < p \leq 0.01$, $0.01 < p \leq 0.05$ and $p > 0.05$ (not significant).

statistical significance, post-hoc tests are carried out using package *lsmeans* to perform pairwise comparison [44].

2.4 Experimental results and analysis

The p-values from the full factorial ANOVA are presented in Table 2.2.

2.4.1 Safety: number of collisions and minimal distance to an obstacle

In this section, the results related to the safety of teleoperation are presented through two metrics: the number of collisions and the mean distance from the UAV to the nearest obstacle.

The number of collisions for the different combinations of scenarios and methods are shown in Figure 2.14. In order to build the model for N_{col}, a Poisson distribution was used. The full factorial ANOVA reveals a significant contribution of both methods and scenarios, as well as an interaction between the two. Post-hoc test shows that both types of feedback offer a statistically significant reduction in the number collision compared to the no feedback case ($p = 0.009$ for m_1 and $p = 0.0001$ for m_2) in the scenario 2. There is no statistical difference between m_1 and m_2 for this scenario ($p = 0.09$). For the other scenarios, the number of collisions is still inferior with both types of feedback (except

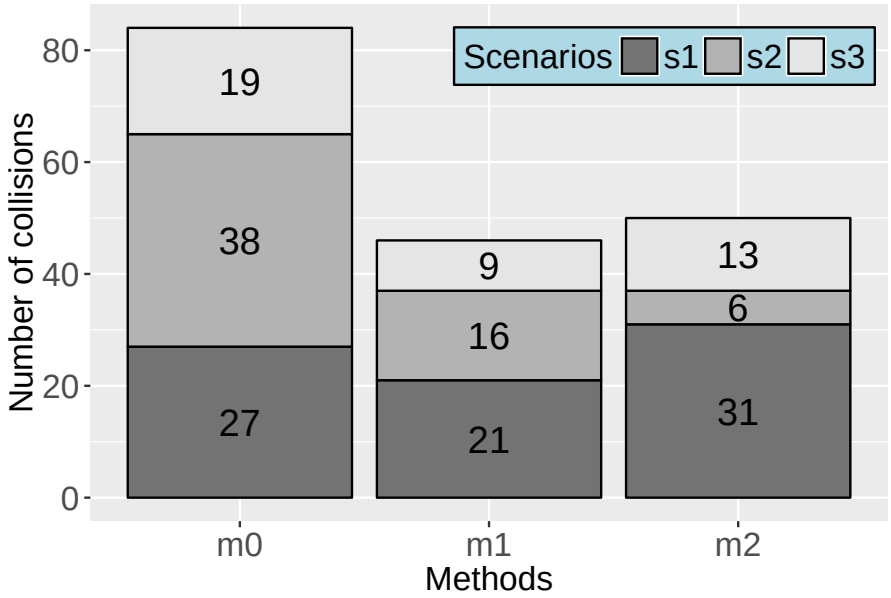


Figure 2.14: Cumulated number of collisions

with m_2 in scenario 1) than without feedback (see Figure 2.14). However, the difference is not statistically significant.

The second metric used to assess safety during the teleoperation is the distance to the nearest obstacle averaged across the run (D_{min}) and is shown in Figure 2.15. As shown in Table 2.2, both methods and scenarios have a significant contribution to D_{min}. Post-hoc analysis reveals that only m_1 incurs a statistically significant increase in D_{min}. While m_2 increases D_{min} on average, this increase is not statistically significant.

2.4.2 Temporal efficiency: time elapsed and average velocity

In this section, the results that are related to the temporal performances of the algorithms are presented through two metrics: the time required to complete the run and the average velocity of the UAV. The average time to complete each run is shown in Figure 2.16 and the average velocity in Figure 2.17.

It can be seen from the ANOVA in Table 2.2 that the feedback type does not have any statistically significant influence on the time elapsed and on the average velocity of the UAV (respectively, $p = 0.45$ and $p = 0.27$).

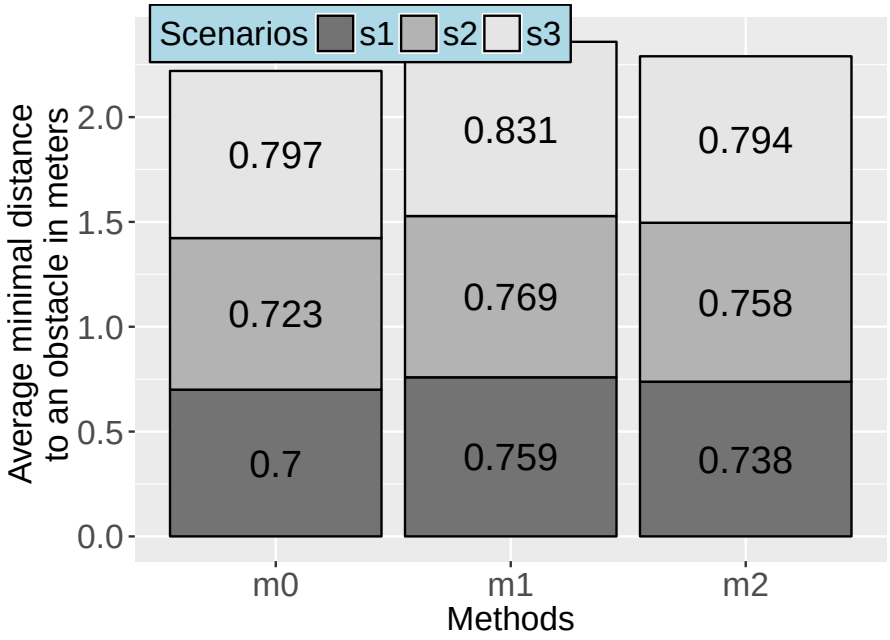


Figure 2.15: Average minimal distance to an obstacle

2.4.3 Workload

A boxplot of the NASA TLX can be found in Figure 2.18. The full test (including pairwise comparison of the factors) was performed. However, since a popular variant of the evaluation does not perform the weighting process [45], the non weighted results are also shown. A meta-study of NASA TLX evaluation has failed to establish a significant difference between the two types of weighting [46]. The trend exhibited by the results is also the same regardless of the weighting method, so the analysis is limited here to the weighted results. The full factorial ANOVA reveals a statistically significant influence of the method on the workload measured by the NASA TLX ($p = 0.029$). Post-hoc analysis shows that m_2 incurs less workload than m_0 with a statistically significant margin ($p = 0.031$). No other statistically significant difference can be found between the other methods.

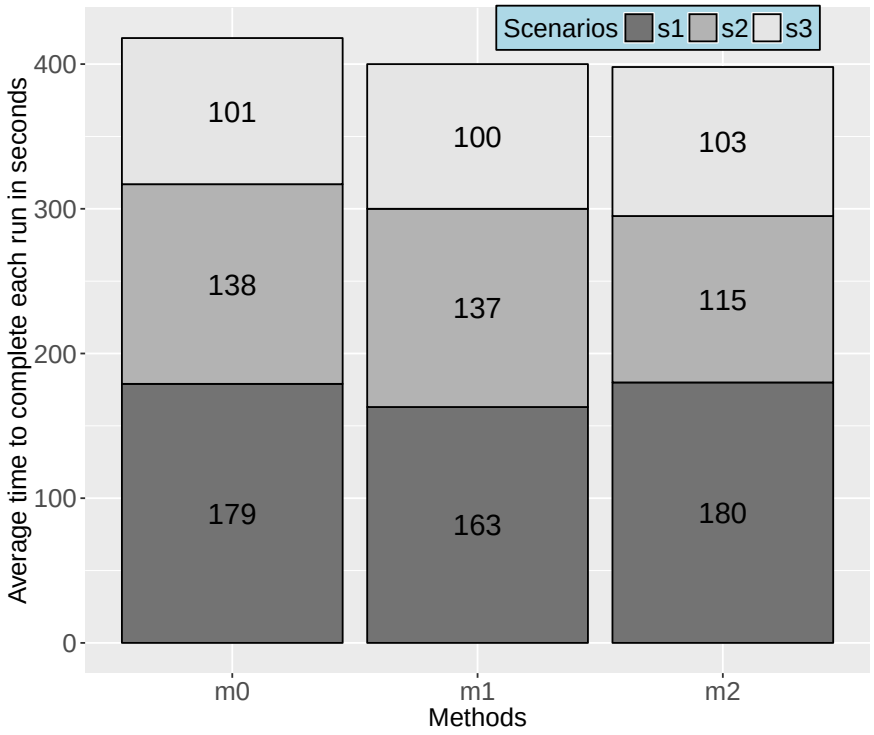


Figure 2.16: Average time to complete each run

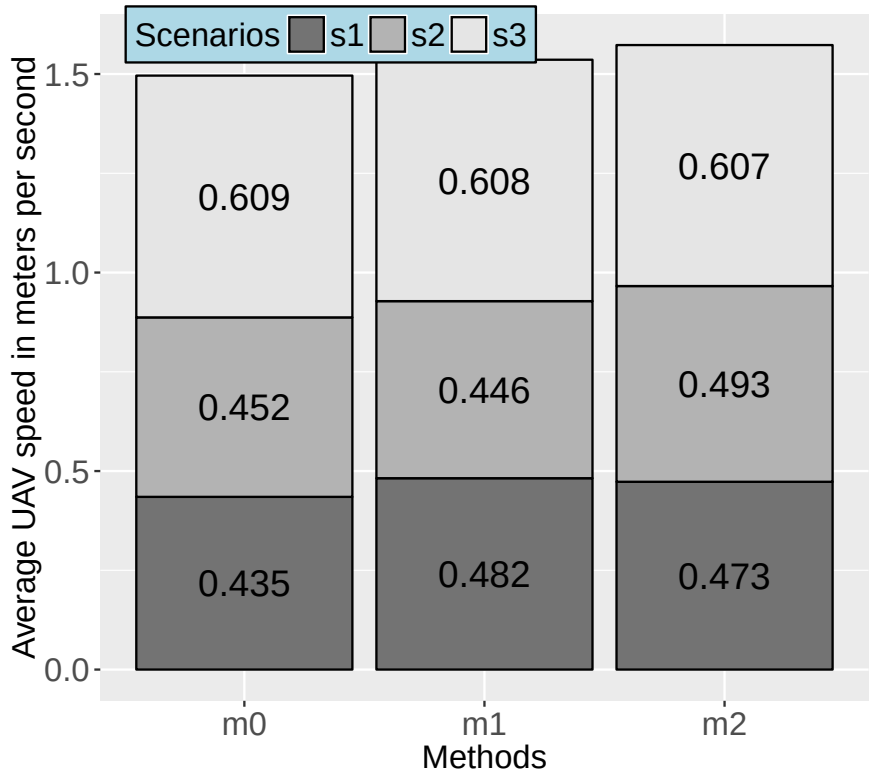


Figure 2.17: Average speed of the UAV

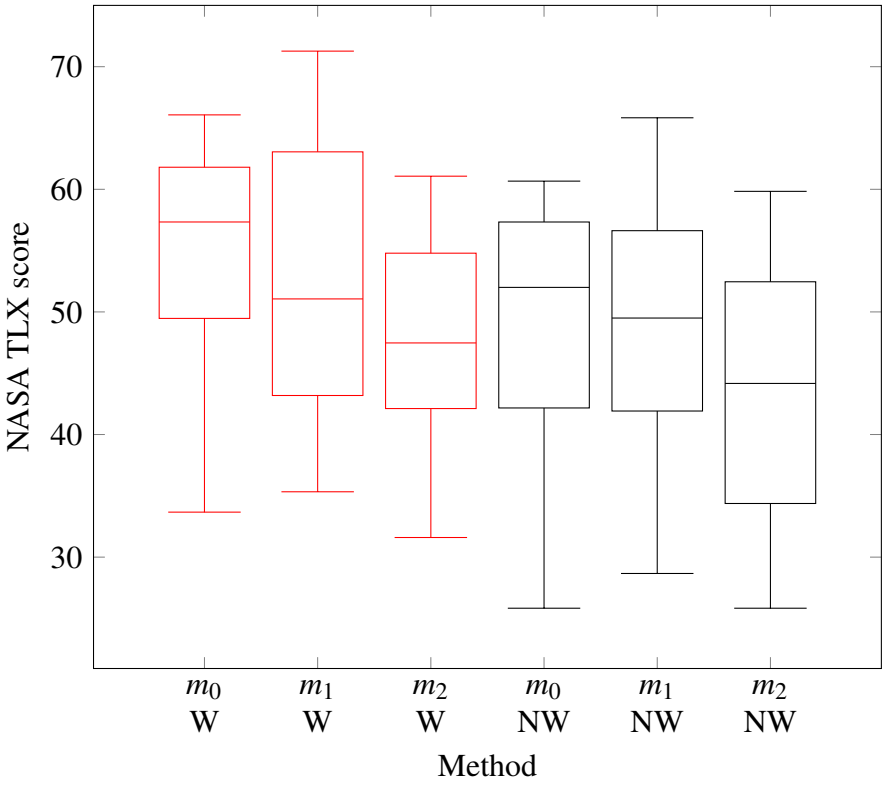


Figure 2.18: Boxplot of the NASA TLX for each method, with weights (W) and without weights (NW). An outlier of 97 for m_0 is not shown on this graph to improve readability

2.4.4 Analysis

Regarding safety, the feedback (both with and without the hole detection scheme) yields a reduction in the number of collision for the second scenario, which involves several windows and holes to go through. It can be seen from Figure 2.14 that the only case where the number of collision was higher with haptic feedback is the first scenario with m_2, although this difference is not statistically significant. That could be explained by the reduced feedback generated by m_2 when facing narrow obstacles, which would increase the number of collisions compared to m_1. This is confirmed by the analysis of the distance to the nearest obstacle, which is statistically longer for m_1, but not for m_2. Since the algorithm used is based on potential fields, the distance to the nearest obstacle is expected to be maximised. Given that m_2 results in a force feedback that is less strong, this result is in line with our expectations. In the third scenario, the small decrease in the number of collision can be explained by the simplicity of the obstacles: the participants did not

find the reduced luminosity to be a real difficulty, defeating the purpose of putting simple obstacles in this scenario. This experiment confirms the hypothesis that haptic feedback can increase the safety of teleoperation in complex environments, however, this safety bonus can decrease if the feedback force is reduced. The case of scenario 2 with m_2 also shows that a stronger feedback can cause collisions when narrow passages are present: our hypothesis is that those collisions are due to the conflict between the feedback and what the operator wants. The exact reasons leading from those conflicts to collisions are unclear, but participants comments' highlight a potential issue: false positives make the operator lose confidence in the feedback and fight more against it. This is the reason the hole detection scheme was added. According to this experiment, this scheme appears to reduce safety compared to the standard feedback algorithm.

The analysis of temporal efficiency does not show any significant difference between the three methods. Given that the time during which the UAV is frozen does not count in the completion time, the time taken can be seen as a measurement of trajectory quality. This experiment shows that the haptic feedback does not result in a quicker completion. A possible explanation is, again, a conflict between the haptic feedback and the operator: on one hand, the feedback helps preventing collisions, which should reduce the time of completion, but on the other hand, the penalty in manoeuvrability results in a longer completion time.

The NASA TLX scores reveal the upside of the hole detection scheme: scenarios where m_2 was used resulted in a less important workload compared to the absence of feedback, which is not the case for m_1. This can be explained by less conflict between the operator and the controller due to the "virtual obstacles" created by m_1 (see Section 2.2.3). However, it can be noted that m_1 slightly decreased the workload compared to m_0, which means that even without the detection scheme, the workload did not increase. This surprising result could be attributed to the fact that less collisions would contribute to reducing the workload: in the experiment, participants are fully aware of each collision, thus a decrease in the number of collisions would be likely to generate less frustration and

a better performance feeling for the participants, both of which are criteria of assessment in the NASA TLX.

This experiment illustrates a trade-off for this algorithm: safety can be increased with a stronger force, but doing so increases workload and reduces manoeuvrability. It was found that in one particular case, the stronger force even decreased safety. This can be attributed to manoeuvrability: a feedback that is too restrictive can lead to mistrust from the operator.

At this points, it is interesting to compare with similar human experiments of UAV teleoperation with different haptic feedback algorithms in the literature. The force-stiffness feedback used for 2D movement of the UAV in the work of Lam et al. [31] results in a reduction (but not a disappearance) of the number of collisions. The workload was reduced by the force-stiffness feedback, but the force feedback alone did not modify the workload. The time elapsed in the run without collisions was generally higher with the haptic feedback. A comparison of three different haptic algorithms [29] found that the TTI was able to reduce the number of collisions while the spring algorithm increased this number. No difference in time elapsed was found. No reduction of the workload was found, but one of the algorithm (the virtual spring) caused an increase in workload. Finally, a study coupling audio feedback with two different haptic algorithms (TTI and VSODDS) [32] found that VSODDS was able to reduce significantly the number of collisions, but not the TTI. No significant workload change was caused by the haptic feedback in this experiment. The TTI increased the time of completion, but the VSODDS did not significantly modify this time. It can thus be concluded that the results from the performed experiment are coherent with those studies [29, 31, 32]: the haptic feedback generally increases safety, however a reduction of workload depends on the algorithm used and its tuning. The time of completion is either increased [31] or unchanged compared to the no feedback case [29, 32].

2.4.5 Limitations and possible improvements

A drawback of using artificial force fields is the need to define constants related to the size of the PRF. This means that the feedback depends only on the spatial configuration around the UAV, and not on the control commands sent to the UAV. For instance, in a situation similar to the second scenario where the UAV has to go through a narrow window, the UAV is necessarily getting closer to an obstacle. Depending on the size of the PRF, this can generate a feedback. There is thus a trade-off between safety and agility: a large PRF means a safe operation, but it is more difficult to manoeuvre in closed spaces, whereas a small PRF makes collisions more likely. This is apparent in the number of collisions and time elapse metrics of our experiment.

A second problem is the exclusive usage of the haptic feedback to perform obstacle avoidance: in the impedance framework, the only way to prevent a movement of the UAV is to prevent the corresponding movement of the end effector. Moreover, the position of all the obstacles has to be reduced to a single vector with three components, meaning that it is difficult to obtain a feedback that is precisely adapted to the location of the obstacles. Combined with the lack of flexibility described in the paragraph above, this means that it can be physically difficult for the operator to achieve the wanted displacement. Moreover, this also means that the operator has complete control over the UAV. While this is fundamentally a positive feature, it also means that errors are possible: in the experiment, the haptic feedback did decrease the number of collisions, but this number is still not zero. This characteristic is not unique to the proposed method, but shared by several haptic feedback algorithms [29, 31, 32]. Since the small UAVs considered in this thesis are very fragile, a single collision can have dire consequences for the UAV and its environment.

A solution to those two problems would be to filter the user's input according to the position of the surrounding obstacles to authorise or reject the movement. This means that obstacle avoidance is now more flexible since there is no manoeuvrability penalty for being close to an obstacle anymore. Moreover, such a method would allow the possibility of flight assistance: should the desired movement be impossible, an alternative displacement

could be proposed that would be coherent with the desired movement. This is a difference with the method proposed in this chapter which only aims to prevent a collision, causing the second problem described above. In other words, assisting the operator might be more efficient than simply preventing collisions.

The drawback is that such a solution requires a more precise representation of obstacles, since the trajectory of the UAV has to be considered obstacle free. Indeed, the use of the PRF means that the algorithm described in this chapter performs well even if only a few points are detected per obstacle. This is shown in the experiment, that features a simulated LIDAR with low vertical resolution. On the other hand, when investigating whether the UAV would collide with a given obstacle, an estimation of the shape of the obstacle is required. In order to solve this issue, a map of the environment surrounding the UAV is required, which is the subject of Chapter 3. Moreover, from a point of view of the sensors, there are two solutions to detect obstacles in all directions around the UAV: either to use several sensors to cover all angles or to use a map (the single sensor simulated in the experiment is unrealistic since it would be occluded by the body of the UAV).

2.5 Conclusion

In this chapter, a new 3D haptic algorithm based on artificial force fields is presented for obstacle avoidance of UAVs. This algorithm is tested in a simulation environment with 11 participants. Scenarios take place indoor in closed spaces. It was found experimentally that compared to the absence of feedback, the proposed base method (m_1) is helpful to avoid obstacles when narrow passages are present. A scheme to reduce the intensity of the feedback when it was not needed (m_2) improved the workload for the operator, at the cost of a slight decrease in safety compared to the basic feedback. Considering the fragility of a UAV, further improvement to the safety would be beneficial. Enhancing the manoeuvrability of the UAV, particularly near obstacles, is another area of improvement. Both could be achieved through the use of a map.

Chapter 3

3D mobile mapping in real time

Contents

A more precise obstacle avoidance algorithm requires a precise estimation of the location and shape of the obstacle. This can be achieved with a mapping method: observing an obstacle over several LIDAR scans provides more information about it. Having precise location for an obstacle would allow to determine safe and dangerous positions for the UAV, increasing safety. Since the movement of the UAV is not restricted, this map should be able to move with the UAV. Thus, in this chapter, the creation of a real time mobile 3D map is investigated.

3.1 Introduction

Mapping, which means creating a representation of the space around a robot, is one of the most important problems in robotics [47]. In the context of obstacle avoidance, a map is not strictly necessary, as illustrated in Chapter 2. It can be enough to have a single point cloud representing the obstacles around the UAV. A map, however, brings several advantages over a single set of measurements:

1. Obstacles that are not directly detected by the sensor are still taken into account, as there is a memory of obstacle locations. This is particularly important since currently, several sensors are needed for complete coverage around the UAV.

2. For a given geometry of the environment, a change in position of the sensor can result in a different point cloud, especially for sensors with sparse outputs such as LIDARs. Using a map can result in a more consistent output since each scan refines the shape of the obstacle.

3. A map can facilitate odometry computation.

A challenge inherent in map creation is the computation of the motion of the robot between each update of the map.

3.1.1 Objectives

For the reasons stated above, it is desirable to get a map of the environment. Such a map should possess several properties:

1. It has to be in 3D, meaning obstacles are detected in all directions of space.

2. It has to be computed in real time. This requirement is flexible, since the runtime is heavily dependent on hardware and code optimisation. The targeted order of magnitude is a few Hertz.

3. It has to adapt to changes in the environment.

4. It should allow computation of the pose of the robot since no mapping is possible if the displacement of the robot between two scans is not known.

5. It has to be adapted to a moving robot, thus scalable. Since the movement of the robot is unlimited, this means that a static map will run out of memory.

3.1.2 The mapping problem : SLAM

The position of a robot is linked to the map created by this robot. In order to know exactly where the obstacles are, it is necessary to know where the robot is, but to know where the robot is, it is needed know where the obstacles are. Thus, the task of building a map is intrinsically linked to the localisation problem. The study of this issue goes back 30 years ago, and a class of methods solving this problem is labelled as Simultaneous Localisation And Mapping (SLAM). A recent overview of SLAM methods is available by Cadena et al. [48]. A SLAM approach involves a state vector that comprises both the position

of the robot and the position of the landmarks observed by the robot. The state vector is updated after each measurement, meaning that both the position of the robot and the position of the landmarks are optimised [49, 50]. The interesting feature of SLAM is the loop closure: when a robot goes back to landmarks already observed, SLAM allows to improve the whole map [48] by reducing the accumulated odometry error. Without loop closures, SLAM is reduced to an odometry scheme.

In the context of the thesis, the problem to tackle is obstacle localisation, and not SLAM in itself. Thus, a solution oriented toward obstacle representation is investigated and obstacle representation is separated from odometry computation. While an odometry solution is required, it will be chosen for its compatibility with a good obstacle representation. It should be noted that feature based SLAM do not provide a compact representation of the obstacles since those are represented by 3D points. On the other hand, there exists dense SLAM algorithms dealing with more complete representation of the environment [48]. They are however quite computationally expensive: real time can be achieved, but with a GPU [51].

For those reasons, the focus of this chapter is obstacle representation. The odometry will be derived from this representation.

3.1.3 Obstacle representation and odometry

A widely used method to represent the environment is the occupancy grid map, developed around 1980 [52, 53]. This method requires the pose of the robot. The occupancy grid map tries to answer the question: given a point in space, is there an obstacle at this point? The method works by subdividing the space into a regular grid. When a new sensor measurement is available, an inverse sensor model is used to modify the probability of each cell to be occupied. The occupancy grid map is thus probabilistic. The original occupancy grid map was developed in 2D. There are several solutions to extend it to 3D. One of them is the elevation map, which uses a 2D grid and adds an associated height for each cell (e.g., [54]). Although efficient in memory, this approach only allows a

single point for a whole spatial column. This makes structures like windows difficult to represent. This is a problem for UAVs, since flying robots in indoor environments might encounter more of those structures compared to ground robots. In order to relax this assumption, the multi-level surface map was developed [55] as an extension of elevation maps, and then extended to probabilistic multi-level maps [56] to include more sources of uncertainty. However, those schemes are not suitable for a dynamic map since they assume purely static obstacles. Moreover, runtime information is missing to evaluate the suitability for online use.

With the above requirements in mind, the occupancy grid map can be adapted to 3D although it then requires a lot of memory. Even if the memory scaling remains poor, especially in 3D, the memory required has an upper bound for a fixed map size. It can also be updated in real time, depending on the sensor model, and can adapt to dynamic environments. A popular implementation of occupancy maps is Octomap [57], which uses octrees to efficiently provide multi-resolution support. Despite this multi-resolution capability, Octomap operates under the assumption that a cell is either fully occupied, or fully empty. This means that to provide an accurate representation of the world, the size of a single cell of the map has to be smaller than a typical feature of the environment.

A solution to use fewer cells is the Normal Distribution Transform (NDT), introduced in the 2D case [58], and extended to 3D by Magnusson, Lilienthal, and Duckett [59]. The NDT operates on a point cloud by computing the mean and covariance of the points inside each spatial cell. NDT is attractive for several reasons:

- The points from the point cloud are not directly kept in memory. Only 11 parameters are required per occupied cell.
- NDT representation can be used to retrieve the odometry. The direct registration of two NDT models is described in [60] and analytical derivatives are provided for the given objective function.
- Multi-resolution is available due to the recursive update scheme.

- Compared to standard occupancy mapping, the occupancy value does not represent the probability of the cell being fully occupied, but indicates how consistent with the measurements the Gaussian inside a cell is. This representation allows cells to be partially occupied. This results in a map that can represent the environment more accurately and with less cells that a standard occupancy grid map. However, this also means that standard algorithms that expect a binary representation of the obstacles (i.e. a cell is occupied or empty) have to be adapted.

An experimental study on the accuracy of NDT compared to standard occupancy grid [61] shows that NDT models can be as accurate as occupancy grids of finer resolutions. While the NDT representation was originally used for point cloud registration [59], it has been extended for mapping in the work of Saarinen et al. [21]. The resulting method is NDT Occupancy Map (NDT OM), which introduces the recursive update of the map. Similar to Octomap [57], NDT OM creates a map in a fixed frame. Both of those approaches reduce the memory footprint compared to a traditional occupancy map (Octomap with multi-resolution pruning, and NDT OM by requiring less cells to represent the same scene). However, given a large enough scene, those methods will run out of memory. A fusion between NDTs is proposed by Stoyanov et al. [5], where the new scan is inserted in the map by keeping the grid aligned.

While the NDT framework allows the computation of the odometry, any algorithm using occupancy maps must provide this odometry. Different ways to retrieve the odometry of the robot are examined below.

So far, a LIDAR sensor has been considered. As will be outlined in Chapter 4, it is possible to get a denser point cloud by adding data from stereo images. Considering LIDAR sensor and stereo camera, there is a wide range of methods to compute odometry. Using the images, obtaining a visual odometry is a first possibility. A good overview on the topic is available in Scaramuzza and Fraundorfer [62][63] and multiple implementations can be found [64–66]. While odometry is typically computed from visible images, multi spectral visual odometry has been investigated as well [67, 68].

A second possibility is to register the successive point clouds. Registration in this context means finding the rigid transformation that minimises the alignment error between two successive scans. In a multi modal setup, this might be the preferred method since depth map can be converted to point clouds, but the opposite is not necessarily true (e.g., if the setup lacks cameras). A good survey on robotic point cloud registration was published in 2015 [69] with again implementations available [70–72]. A standard algorithm to register point clouds is the iterative closest point (ICP) algorithm [70]. The original ICP algorithm supposes that successfully matched individual points in consecutive scans have to correspond to the same physical point. The *point-to-plane* variant [73, 74] has been developed to relax this assumption. Since then, numerous variants of ICP emerged [69]. The NDT [58, 72] uses the same general idea as ICP, although with a different representation of the obstacles.

It is important to note that in practice, for a UAV, several sources (IMU, the map, a camera) would be fused to get the final odometry. Not only would this fusion results in more precise odometry, but the odometry would also be computed faster: the controller of the UAV, for instance, needs odometry data and the NDT map alone would be too slow to provide it. An example of an odometry system fusing visual odometry and scan registration to get the ego motion of a UAV is presented in the work of Droeschel et al. [25].

The next section will present in more details the NDT algorithm and occupancy mapping, as well as their combination into NDT OM [21]. Then, a modification of the NDT OM algorithm to make it mobile will be proposed, along with an improvement to the ray tracing method. Finally, testing of the algorithm will be performed on two public datasets.

3.2 Background

In this section, some background is presented on ICP, the NDT and its combination with occupancy mapping. Since the NDT related algorithms are spread over several publica-

tions [5, 6, 21, 59–61, 72, 75], it can be difficult to understand the flow of procedures required to build a complete map. This section aims to present the different techniques in a coherent manner and with details, to have a complete understanding of the whole process, from a set of point clouds, to a map.

Furthermore, a new method is proposed to build up on the NDT OM algorithm[1]. The proposed implementation is described in more details in Section 3.3.

3.2.1 Rigid transforms and coordinate frames

This section describes the conventions used to describe rigid transforms. A rigid transform is described by a four by four matrix $\mathbf{T}$ so that

$$\mathbf{T} = \left[\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right], \tag{3.1}$$

with $\mathbf{R}$ being a three by three rotation matrix (i.e. $\mathbf{R} \in \mathrm{SO}(3)$) and $\mathbf{t}$ a three by one translation vector. The notations $\mathrm{rot}(\mathbf{T})$ and $\mathrm{trans}(\mathbf{T})$ refer to $\mathbf{R}$ and $\mathbf{t}$ respectively. If i and j are two reference frames, the rigid transform $\mathbf{T}^{i\to j}$ sends an element from reference frame i into j, or in other word, if $\mathbf{x}_i$ is a homogeneous vector expressed in frame of reference i, and $\mathbf{x}_j$ is expressed in j, the following equality holds:

$$\mathbf{T}^{i\to j}\mathbf{x}_i = \mathbf{x}_j. \tag{3.2}$$

Each transform can be time indexed. For two times t_1 and t_2, the rigid transform $\mathbf{T}^{i\to j}_{t_1\to t_2}$ is defined as:

$$\mathbf{T}^{i\to j}_{t_1} = \mathbf{T}^{i\to j}_{t_2}\mathbf{T}^{i\to j}_{t_1\to t_2}. \tag{3.3}$$

If $\mathcal{P}$ is a point cloud composed of points $\mathbf{x}_i$ expressed in homogeneous coordinates

[1] `https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic`

and $\mathbf{T}$ is a rigid transform, the notation $\mathbf{T}\mathcal{P}$ designates a new point cloud composed of the points $\mathbf{T}\mathbf{x}_i$.

Three reference frames are used:

- the world reference frame, which represent the outside world (denoted w),
- the map reference frame (denoted m). Note that, by definition of the proposed algorithm, the rigid transform between the map reference frame and the world reference frame can only be a translation, since the map is never rotated with respect to the world reference frame.
- the sensor reference frame, aligned with the sensor that acquires the localisation data (denoted s).

Note that this way, the rotation in $\mathbf{T}^{m\to w}$ is always the identity matrix.

3.2.2 Basic ICP

For clarity purposes, a basic version of the ICP algorithm is described here since the general idea of this algorithm is the same as the NDT. Given two point clouds $\mathcal{P}_1$ and $\mathcal{P}_2$, and an initial rigid transformation $\mathbf{T}_{\text{init}}^{1\to 2}$, the algorithm aims to find the final rigid transformation between the two point clouds $\mathbf{T}^{1\to 2}$. The algorithm is described in Algorithm 3.1.

The algorithm operates by incrementally minimising the error between each associated points inside a pair. After a refinement to the rigid transform is found, it is added the current rigid transform. The process then goes on until the error is small enough. The challenge mostly lies in the point association part: given a point in the first point cloud, which is the corresponding point in the second point cloud? In the original algorithm, the points that are the closest are associated together. As presented here, the algorithm relies on the fact that there is a full overlap between the two point clouds, which is not guaranteed in practice. Thus, it is necessary to filter outliers and apply a weight to each pair in the error function. It is also possible to use other strategies such as point-to-plane

Algorithm 3.1: The ICP algorithm

Input : two point clouds $\mathcal{P}_1$ and $\mathcal{P}_2$,
an initial rigid transformation $\mathbf{T}_{\text{init}}^{1\to2}$.
Output: rigid transformation between the two point clouds $\mathbf{T}^{1\to2}$.

$\mathbf{T}^{1\to2} = \mathbf{T}_{\text{init}}^{1\to2}$; `// Initialise the rigid transformation`
do
  $\mathcal{P} = \mathbf{T}^{1\to2}\mathcal{P}_1$;
  `// Associate each point` $\mathbf{p}_i^1$ `of` $\mathcal{P}$ `with a point` $\mathbf{p}_j^2$ `of` $\mathcal{P}_2$,
  `creating a set` $\mathbf{S}$ `of pairs` $\left(\mathbf{p}_i^1, \mathbf{p}_j^2\right)$
  $\mathbf{S} = \varnothing$;
  for *each* $\mathbf{p}_i^1 \in \mathcal{P}$ **do**
    Find a match $\mathbf{p}_j^2$ for $\mathbf{p}_i^1$;
    $\mathbf{S} \leftarrow \left(\mathbf{p}_i^1, \mathbf{p}_j^2\right)$;
  end
  $\varepsilon\left(\mathcal{P}, \mathcal{P}_2\right) = \sum_{\mathbf{S}\in\left(\mathbf{p}_i^1, \mathbf{p}_j^2\right)} \|\mathbf{p}_i^1 - \mathbf{p}_j^2\|$; `// Compute the error`
  $\mathbf{T}^{1\to2} = \text{argmin}_{\mathbf{T}^{1\to2}}\left(\varepsilon\left(\mathcal{P}, \mathcal{P}_2\right)\right)$; `// Minimise this error to get a partial transformation`
while ε *is too large*;
return $\mathbf{T}^{1\to2}$;

association [73, 74], that minimises the error along the normal of the surface only. To go further, Generalized-ICP [71] considers that each point is generated from a Gaussian distribution, centred on the observation. This can be viewed as plane-to-plane matching.

The association stage can be speeded up by the use of KD-Trees [76] to find the closest point, giving the point association a computational complexity of $\mathcal{O}(n \log n)$.

3.2.3 The 3D Normal Distribution Transform (NDT)

The general structure of the NDT algorithm is similar to ICP: the principle is still to incrementally refine the rigid transformation by minimising an error metric computed over both scans. The difference is in the representation: instead of dealing with the raw points, the NDT uses another representation of the scan, a set of Gaussian probability distributions.

The NDT was first developed in 2D [58], then extended to 3D [72]. A remarkable advantage of the NDT is the odometry estimation that can be derived from it [59, 61].

However, in this thesis, its main interest lies in its obstacle representation model.

The first step in the NDT pipeline is to divide the space in regular cubic cells. Let a point cloud $\mathcal{P}$ composed of points $\mathbf{p} \in \mathbb{R}^3$ be the output of the sensor. Then for each cell $\mathcal{C}_i$ containing $|\mathcal{C}_i|$ points (more than a fixed threshold), the mean and covariance of the points inside the cell are computed as shown in equations (3.4) and (3.5).

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{p} \in \mathcal{C}_i} \mathbf{p}, \tag{3.4}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{|\mathcal{C}_i| - 1} \sum_{\mathbf{p} \in \mathcal{C}_i} (\mathbf{p} - \boldsymbol{\mu}_i)(\mathbf{p} - \boldsymbol{\mu}_i)^T. \tag{3.5}$$

For a given point $\mathbf{x}$, the probability to belong to the obstacle in cell $\mathcal{C}_i$ is defined by a normal distribution: $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. Let the set of Gaussian distributions computed from a point cloud $\mathcal{P}$ be called $\mathrm{NDT}(\mathcal{P})$. As such, a set $\mathrm{NDT}(\mathcal{P})$ composed of n cells can be defined as:

$$\mathrm{NDT}(\mathcal{P}) = [(|\mathcal{C}_1|, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \cdots, (|\mathcal{C}_n|, \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)]. \tag{3.6}$$

In the following, the number of occupied cells in an NDT is written $|\mathrm{NDT}(\mathcal{P})|$.

In the original 3D NDT [59], a new point cloud is matched against an NDT representation. The registration method has been improved by matching the NDTs representation directly [60]. This modified version, coined D2D NDT (Distribution to Distribution NDT) was found to be more precise and faster than the original version [60]. This is the matching method adopted in the proposed algorithm. In addition, a more in depth examination of the D2D NDT has been carried out by Stoyanov et al. [75], describing the rotation representation problem and a way to estimate the covariance of the odometry based on a more accurate estimation of the variance of the objective function considering the noise of the input [77].

As described in the D2D NDT [60], given an NDT model $\mathrm{NDT}(\mathcal{P})$, the likelihood

that a point $\mathbf{x} \in \mathbb{R}^3$ is generated from this model is:

$$p\left(\mathbf{x}|\mathrm{NDT}\left(\mathcal{P}\right)\right) = \sum_{i=1}^{|\mathrm{NDT}(\mathcal{P})|} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i). \tag{3.7}$$

Then, if $\mathbf{T}$ is a rigid transformation, characterized by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$, the transformation of an NDT by $\mathbf{T}$ is defined as:

$$\mathbf{T}\left(\mathrm{NDT}\left(\mathcal{P}\right)\right) = \left[\left(\left|\mathcal{C}_1\right|, \mathbf{T}\left(\boldsymbol{\mu}_1\right), \mathbf{T}\left(\boldsymbol{\Sigma}_1\right)\right), \cdots, \left(\left|\mathcal{C}_n\right|, \mathbf{T}\left(\boldsymbol{\mu}_n\right), \mathbf{T}\left(\boldsymbol{\Sigma}_n\right)\right)\right], \tag{3.8}$$

where

$$\mathbf{T}\left(\boldsymbol{\mu}_i\right) = \mathbf{R}\boldsymbol{\mu}_i + \mathbf{t}, \tag{3.9}$$

$$\mathbf{T}\left(\boldsymbol{\Sigma}_i\right) = \mathbf{R}\boldsymbol{\Sigma}_i\mathbf{R}^t. \tag{3.10}$$

$$\tag{3.11}$$

From there, as described in [75], it is possible to define a distance between two NDTs based on the L_2 norm for a given rigid transform between the NDT models:

$$D_{\mathcal{P}_1,\mathcal{P}_2}\left(\mathbf{T}\right) = \int \left(p\left(\mathbf{x}|\mathrm{NDT}\left(\mathcal{P}_1\right)\right) - p\left(\mathbf{x}|\mathbf{T}\left(\mathrm{NDT}\left(\mathcal{P}_2\right)\right)\right)\right)^2 \mathbf{dx} \tag{3.12}$$

$$\begin{aligned} &= \int \left(p\left(\mathbf{x}|\mathrm{NDT}\left(\mathcal{P}_1\right)\right)\right)^2 \mathbf{dx} + \int \left(p\left(\mathbf{x}|\mathbf{T}\left(\mathrm{NDT}\left(\mathcal{P}_2\right)\right)\right)\right)^2 \mathbf{dx} \\ &\quad - 2\int \left(p\left(\mathbf{x}|\mathrm{NDT}\left(\mathcal{P}_1\right)\right) p\left(\mathbf{x}|\mathbf{T}\left(\mathrm{NDT}\left(\mathcal{P}_2\right)\right)\right)\right) \mathbf{dx}. \end{aligned} \tag{3.13}$$

The first two terms in equation (3.13) are constant with regard to $\mathbf{T}$ since the rigid transformation of an NDT model only shifts and rotates the Gaussian distributions. Thus an integration over $\mathcal{R}^3$ is not affected by this transform. Then, the distance between two

NDTs is reduced to the cross term and a constant C :

$$D_{\mathcal{P}_1,\mathcal{P}_2}(\mathbf{T}) = C - 2\int \left(p\left(\mathbf{x}|\mathrm{NDT}\left(\mathcal{P}_1\right)\right) p\left(\mathbf{x}|\mathbf{T}\left(\mathrm{NDT}\left(\mathcal{P}_2\right)\right)\right)\right)\mathbf{dx}. \tag{3.14}$$

Injecting equation (3.7) and noting $|\mathrm{NDT}(\mathcal{P}_1)| = n_1$ and $|\mathrm{NDT}(\mathcal{P}_2)| = n_2$ yields:

$$D_{\mathcal{P}_1,\mathcal{P}_2}(\mathbf{T}) = C - 2\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\int \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mathcal{N}(\mathbf{x}|\mathbf{T}\left(\boldsymbol{\mu}_j\right),\mathbf{T}\left(\boldsymbol{\Sigma}_j\right))\mathbf{dx}. \tag{3.15}$$

The following identity is now used:

$$\int \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)\mathbf{dx} = \mathcal{N}(\mathbf{0}|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j,\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j) \tag{3.16}$$

This identity is proposed in [78], but is not proved, so a proof is included in Appendix B.1.

Using identity (3.16) in equation (3.15) yields:

$$D_{\mathcal{P}_1,\mathcal{P}_2}(\mathbf{T}) = C - 2\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\mathcal{N}(\mathbf{0}|\boldsymbol{\mu}_i - \mathbf{T}\left(\boldsymbol{\mu}_j\right),\boldsymbol{\Sigma}_i + \mathbf{T}\left(\boldsymbol{\Sigma}_j\right)) \tag{3.17}$$

Thus, the objective function to minimise in order to get the rigid transform is, with positive real tuning parameters d_1 and d_2 added:

$$f_{obs}(\mathbf{T}) = -\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} d_1 \exp\left(-\frac{d_2}{2}\left(\mathbf{R}\boldsymbol{\mu}_j + \mathbf{t} - \boldsymbol{\mu}_i\right)^t \left(\boldsymbol{\Sigma}_i + \mathbf{R}\boldsymbol{\Sigma}_j\mathbf{R}^t\right)^{-1}\left(\mathbf{R}\boldsymbol{\mu}_j + \mathbf{t} - \boldsymbol{\mu}_i\right)\right). \tag{3.18}$$

More details on the tuning parameters d_1 and d_2 can be found in the work of Stoyanov, Magnusson, and Lilienthal [60]. The derivatives of f_{obs} can be computed analyti-

cally [75]. The rigid transform that realises the minimisation of f_{obs} is called $\mathbf{T}^{\Delta}$. Note that, by definition, the Gaussian distributions from different cells of the same NDT should be disjoint. Thus it is reasonable, for a given cell $\mathcal{C}$, to limit the computation of the exponential in equation (3.18) to the cells close to $\mathcal{C}$. The standard approach is to use only the closest cell [60] or a neighbourhood around the centre cell [5]. This results in reduced computational requirements. In other words, the double sum in equation (3.18) is not actually computed, because the association only considers cells close enough for the exponential to be high enough.

In order to get a better, but slower, registration result, it is possible to use the multi resolutions method described in the original point to distribution matching [59]. The idea is to build several NDTs from the same scan, but with different cell sizes (i.e. different resolutions). Then, the first registration is performed with the lowest resolution, and the result is fed as an initial guess to the next registration, between the NDTs with a higher resolution. The process is repeated until the highest resolution is reached.

Now that the registration method has been explained, several advantages of 3D NDT can be mentioned:

- 3D NDT uses analytical derivatives of the objective function.
- The covariance matrix is computed directly from the point cloud, and does not require any additional assumption such as planarity.
- The algorithm operates on the NDT representation directly, not on the point cloud, which reduces the number of operations. For the same reason, the memory scaling is better compared to an algorithm operating directly on the point cloud.

A drawback of the NDT is the requirement to build the Gaussians: if few points are available, then the resulting Gaussian can be impossible to compute or a degenerated case. A solution to this problem is proposed by Hong and Lee [79]. The authors use the noise of the LIDAR to define a probability for each point sample. This method increases the precision of the matching process, but slows the algorithm down.

3.2.4 Fusion of two 3D NDT representation

It was shown how to register two NDT representation in the section before. Once the rigid transformation between both point clouds is estimated, a fusion scheme to combine both in a single NDT representation was proposed [21]. The idea is that, once the registration is complete, the new scan can be moved to the world reference frame and then a new NDT can be computed. The new NDT representation, as well as the former one, which will be called *global* in the following, should then have perfectly aligned grids. Thus, for each cell, the mean, covariance, and number of points have to be combined. The chosen method [21] was inspired from a prior work on variance fusion [80] and called Recursive Sample Covariance update (RSC). For two corresponding cells with means $\left(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j\right)$, covariances $\left(\boldsymbol{\Sigma}_i, \boldsymbol{\Sigma}_j\right)$, and number of points $\left(n_i, n_j\right)$, the RSC scheme creates a new cell with mean $\boldsymbol{\mu}_{i+j}$, covariance $\boldsymbol{\Sigma}_{i+j}$, and number of points n_{i+j} so that:

$$\boldsymbol{\mu}_{i+j} = \frac{n_i \boldsymbol{\mu}_i + n_j \boldsymbol{\mu}_j}{n_i + n_j}, \tag{3.19}$$

$$\begin{aligned} \boldsymbol{\Sigma}_{i+j} = \frac{1}{n_i + n_j - 1} \Bigg[& \left(n_i - 1\right) \boldsymbol{\Sigma}_i + \left(n_j - 1\right) \boldsymbol{\Sigma}_j \\ & + \frac{n_i n_j}{n_i + n_j} \left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right) \left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right)^t \Bigg], \end{aligned} \tag{3.20}$$

$$n_{i+j} = n_i + n_j. \tag{3.21}$$

It was shown that the RSC scheme provides results that are quite similar to the one obtained if the new mean and covariance are computed directly from all the actual points in the cell [21]. A weakness of the RSC is the unbounded growth of the quantities multiplied by n_i and n_j (e.g., $n_i \boldsymbol{\mu}_i$ or $\left(n_i - 1\right) \boldsymbol{\Sigma}_i$). In practice, those values will overflow with enough runtime. To solve that, Adaptive Recursive Sample Covariance (ARSC) was developed in the same work [21]. The idea is to define a threshold M_{max} for the number of points. It is considered that above this threshold the influence of n additional points, with $n \ll M_{max}$,

bring small changes to the mean and covariance of the cell. When the number of points in a cell goes above M_{max} by n, the mean and covariance are scaled by $M_{max}/(M_{max}+n)$ and $(M_{max}-1)/(M_{max}+n-1)$ respectively, and the total number of point is then capped at M_{max}.

An interesting thing to note is that this scheme allows multi resolution maps: by keeping an NDT representation at the finest resolution, it is possible to build NDT at lower resolutions. Thus, it is possible to apply multi resolution matching (see Section 3.2.3) to register the NDTs although it was deemed too slow for a real time implementation by Stoyanov et al. [5].

3.2.5 Obstacle representation with Occupancy Mapping

Since the NDT representation relies on dividing spaces into regular cubic cells and computing the odometry, it looks tempting to combine it with occupancy maps. The original occupancy map concept [52, 53] is to divide the space in regular cells. Each cell has two possible states: occupied or free. The probability for a cell to be occupied is updated with each new measurement. The basic occupancy mapping theory is described in this section, a more detailed coverage can be found in the book of Thrun [81]. It is followed by the adaptation of occupancy mapping to the NDT representation [21].

The notation $\mathcal{C}_i$ is still used to describe a cell, m is the map (i.e. the collection of all the $\mathcal{C}_i$), $\mathbf{Z}_{1:t}$ is the set of measurements from the initial time to time t (a set of measurements $\mathbf{Z}_t$ at time t is a set of 3D points $\mathbf{z}_t^i$), and $\mathbf{l}_{1:t}$ is the set of all the poses of the robot. The ultimate objective is to compute the posterior probability:

$$p\left(m|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right). \tag{3.22}$$

In other words, the objective is to update the probability for a cell to be occupied, for all cells of the map. In this section, this update is formulated using the log odd notation.

In order to make the problem numerically solvable, it is necessary to approximate each cell as independent from each other. In practice, this is often imprecise, since an obstacle might span several cells. This approximation was shown by Thrun [81] (chapter 9.4) to be valid only for sensors that have a narrow field of view for a given obstacle (e.g., a LIDAR).

With this approximation, the posterior probability (3.22) can be written as a product of the probability $p(\mathcal{C}_i)$ that the cell $\mathcal{C}_i$ is occupied:

$$p(m|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}) = \prod_i p(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}). \tag{3.23}$$

Using Bayes' theorem gives us the following (see Appendix B.2 for proof):

$$P(Z|X,Y) = \frac{P(Y|Z,X)P(Z|X)}{P(Y|X)} \tag{3.24}$$

Applying this to the probability of a single cell being occupied $p(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t})$ gives:

$$p(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}) = \frac{p(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t})\,p(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t})}{p(\mathbf{Z}_t|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t})}. \tag{3.25}$$

A Markov assumption stating that, given the state of a cell, the previous observations do not inform us about the current observation gives:

$$p(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t}) = p(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{l}_t). \tag{3.26}$$

This assumption might seem problematic: since a single cell is considered, and not the whole map, this assumption would fail if the sensor observations link several cells.

However, if again, a sensor with a narrow beam is considered, then it is less likely to link several cells (i.e. to suppose an obstacle is present in several adjacent cells from a single measurement). Additionally, the probability of a cell being occupied does not depend on the current position of the robot if only past measurements are considered. Moreover, the probability of a measurement being observed at time t only depends on the position at time t. In other words:

$$p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t}\right) = p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right), \tag{3.27}$$

$$p\left(\mathcal{C}_i|\mathbf{l}_t\right) = p\left(\mathcal{C}_i\right). \tag{3.28}$$

Using equation (3.26) and (3.27) in equation (3.25) yields:

$$p\left(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right) = \frac{p\left(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{l}_t\right) p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}{p\left(\mathbf{Z}_t|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t}\right)}. \tag{3.29}$$

Using equation (3.24) yields:

$$p\left(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{l}_t\right) = \frac{p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right) p\left(\mathbf{Z}_t|\mathbf{l}_t\right)}{p\left(\mathcal{C}_i|\mathbf{l}_t\right)}. \tag{3.30}$$

Inserting equation (3.30) into equation (3.29) and using equation (3.28) to simplify the denominator gives:

$$p\left(\mathcal{C}_i|\mathbf{Z}_{1:t},l_{1:t}\right) = \frac{p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right) p\left(\mathbf{Z}_t|\mathbf{l}_t\right) p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}{p\left(\mathcal{C}_i\right) p\left(\mathbf{Z}_t|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t}\right)}. \tag{3.31}$$

Now, the same can be done for the opposite event (i.e. the cell $\mathcal{C}_i$ is empty). Let us

call this probability $p\left(\overline{\mathcal{C}_i}\right)$, using the same reasoning yields:

$$p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right) = \frac{p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_t,\mathbf{l}_t\right)p\left(\mathbf{Z}_t|\mathbf{l}_t\right)p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}{p\left(\overline{\mathcal{C}_i}\right)p\left(\mathbf{Z}_t|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t}\right)}. \tag{3.32}$$

The ratio of equation (3.31) and equation (3.32) gives:

$$\begin{aligned}\frac{p\left(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right)}{p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right)} &= \frac{p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right)p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)p\left(\overline{\mathcal{C}_i}\right)}{p\left(\mathcal{C}_i\right)p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_t,\mathbf{l}_t\right)p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}, \\ &= \frac{p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right)}{p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_t,\mathbf{l}_t\right)}\frac{p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}{p\left(\overline{\mathcal{C}_i}|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}\frac{p\left(\overline{\mathcal{C}_i}\right)}{p\left(\mathcal{C}_i\right)}, \\ &= \frac{p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right)}{1-p\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right)}\frac{p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}{1-p\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right)}\frac{1-p\left(\mathcal{C}_i\right)}{p\left(\mathcal{C}_i\right)}.\end{aligned} \tag{3.33}$$

Equation (3.33) gives a recursive and efficient way to compute the probability of occupancy of a given cell. In order to avoid discontinuities when probabilities are near 0 or 1 and avoid multiplying small floating point numbers, the log odds notation is used:

$$l_{\text{odd}}\left(x\right) = \log\left(\frac{p\left(x\right)}{1-p\left(x\right)}\right). \tag{3.34}$$

A plot of the l_{odd} function is provided in Figure 3.1.

Writing equation (3.33) with the l_{odd} function gives:

$$l_{\text{odd}}\left(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t}\right) = l_{\text{odd}}\left(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t\right) + l_{\text{odd}}\left(\mathcal{C}_i|\mathbf{Z}_{1:t-1},\mathbf{l}_{1:t-1}\right) - l_{\text{odd}}\left(\mathcal{C}_i\right) \tag{3.35}$$

Equation (3.35) is the basis of occupancy grid mapping. The last term is a prior probability, which can be set to zero if there is no initial information about the map (i.e. $p\left(\mathcal{C}_i\right) = 0.5$). This will be assumed to be the case in the following. The probability

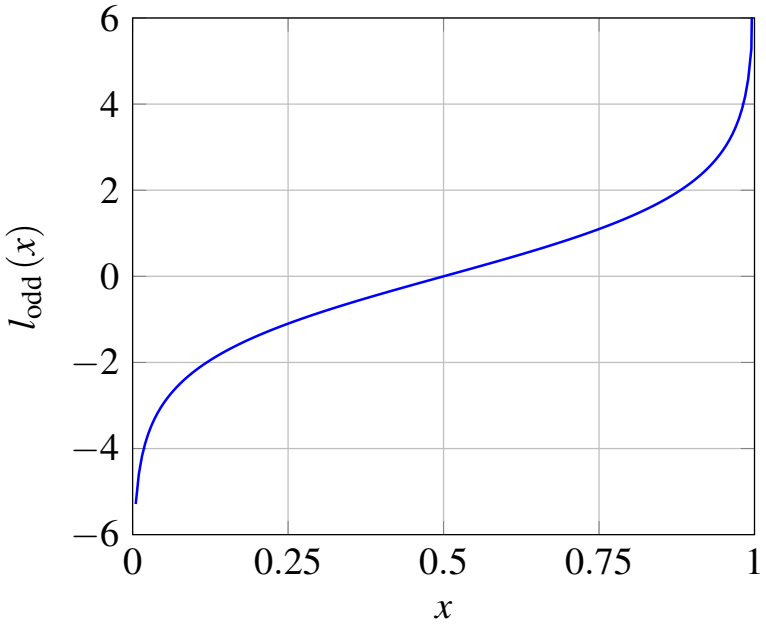


Figure 3.1: l_{odd} function

$p(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t)$ is called the *inverse sensor model*, since it gives the probability of a cell being occupied according to a measurement. A forward sensor model would give the probability $p(\mathbf{Z}_t|\mathcal{C}_i,\mathbf{l}_t)$, which is much more difficult to compute since the space of measurement is bigger than the binary state of occupied or not occupied.

From now on, the probability of a cell being occupied (i.e. $p(\mathcal{C}_i|\mathbf{Z}_{1:t},\mathbf{l}_{1:t})$) will be called the occupancy $\text{occ}_i(t)$. Equation (3.35) gives us the following formula to update the occupancy of a given cell using the log odd function:

$$l_{\text{odd}}(\text{occ}_i(t)) = l_{\text{odd}}(\text{occ}_i(t-1)) + l_{\text{odd}}(\mathcal{C}_i|\mathbf{Z}_t,\mathbf{l}_t). \tag{3.36}$$

There is a problem with this update equation in a dynamic environment. If a cell is repeatedly observed occupied (or empty), the absolute value of $l_{\text{odd}}(\text{occ}_i(t))$ in equation (3.36) will grow unbounded. As a result, if the state of the cell dynamically changes, the log odd of the occupancy value will need to go back to 0 before this change is perceived in the map. In other words, if a cell is observed occupied n times, it will have to be

observed empty n times before the state of the map changes accordingly. This problem has been examined by Yguel, Aycard, and Laugier [82] and a popular solution is to clamp the value of the log odd occupancy [21, 57]. Let K_{occ} be a positive real constant: after updating $l_{\text{odd}}(\text{occ}_i(t))$ with equation (3.36), the following operation is applied to the log odd of the occupancy:

$$l_{\text{odd}}(\text{occ}_i(t)) = \begin{cases} -K_{\text{occ}} & \text{if } l_{\text{odd}}(\text{occ}_i(t)) < -K_{\text{occ}}, \\ K_{\text{occ}} & \text{if } l_{\text{odd}}(\text{occ}_i(t)) > K_{\text{occ}}, \\ l_{\text{odd}}(\text{occ}_i(t)) & \text{otherwise.} \end{cases} \tag{3.37}$$

By doing this, there is effectively a cap on the occupancy. The higher K_{occ} is, the slower the map will adapt to dynamic changes. On the opposite, lower values of K_{occ} will make the map update faster meaning more sensitivity to potential outliers.

3.2.6 Occupancy mapping and NDT

The association between occupancy mapping and NDT was developed in 2013 and resulted in NDT Occupancy Map (NDT OM) [6, 21]. This section aims to explain how this association works.

A standard occupancy grid in 3D is considered, as defined in Section 3.2.5. The same grid is then used to compute an NDT representation, as described in Section 3.2.3. This means that inside a cell, there are 4 elements:

- A log odd occupancy value.
- If the cell is occupied, a mean vector, a covariance matrix and a number of points are defined. Those element describe the spatial probability distribution of an obstacle inside this cell.

There is then a total of 11 numbers to store per occupied cell (1 for occupancy, 1 for the number of points, 3 for the mean, 6 for the covariance).

In this section, a single measurement $\mathbf{z}_t$ is considered, which for a LIDAR sensor would be a 3D point corresponding to the end of the ray, in the set $\mathbf{Z}_t$. As outlined in equation (3.35) an inverse sensor model is needed in order to update the occupancy in each cell, i.e. a probability, for a cell $\mathcal{C}_i$, a measurement $\mathbf{z}_t$, and a pose $\mathbf{l}_t$:

$$p(\mathcal{C}_i|\mathbf{z}_t, \mathbf{l}_t)$$

The inverse sensor model described in the NDT OM [21] uses raytracing. The later computes for each 3D point measurement $\mathbf{z}_t$ the set $\mathcal{S}^t$ of all the cells that are traversed by the ray from the sensor to the measurement. For each cell in $\mathcal{S}^t$, the occupancy needs to be updated. According to equation (3.34), if the occupancy is superior to 0.5, then the value of the log odd occupancy (which is what is actually stored) will increase. In the opposite case, the occupancy will decrease. The difference with NDT OM [21] is that there might be a distribution describing an obstacle inside a cell.

As a reminder, in the standard occupancy theory, there is no Gaussian distribution inside a cell, a cell can only be empty or occupied. This means that if $\mathcal{C}_i \in \mathcal{S}^t$ (i.e. the cell is traversed by the ray), the probability $p(\mathcal{C}_i|\mathbf{z}_t, \mathbf{l}_t)$ in equation (3.36) is a constant. This constant is superior to 0.5 when $\mathbf{z}_t$ is inside $\mathcal{C}_i$ (i.e. the cell is occupied), else inferior to 0.5 (the cell is considered empty, because traversed by the ray).

However, with NDT OM, it is possible to be more precise: a cell can now be partially occupied. Thus, it is possible that a ray passes through a cell containing a Gaussian distribution. In this case, the occupancy can be reduced according to the consistency between the location of the ray inside the cell and the distribution. In other words, if the path of the ray is consistent with the distribution inside the cell, then the log odd occupancy should decrease very little. For instance, if the Gaussian distribution occupies the left part of the cell and the ray goes through the right part of the cell, then the Gaussian position is consistent with the path of the ray. If, however, the path of the ray is not consistent with the distribution inside the cell (e.g., the ray overlaps with the distribution),

then the distribution inside the cell might be erroneous. In this case, the log odd occupancy should more sharply decrease. Erroneous Gaussians might appear for several reasons, including wrong odometry, dynamic environment or a multi modal distribution of points inside the cell.

The strategy to evaluate the consistency between a ray and a Gaussian distribution proposed in the original implementation is as follows [21]: for any point $\mathbf{x}_m$ on the ray, there is a probability $p(\mathbf{x}_m|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ that this point is explained by the distribution inside the cell. There is also a probability $p(\mathbf{x}_m|\mathbf{z}_t)$ that $\mathbf{x}_m$ can be explained by the measurement $\mathbf{z}_t$. In order to evaluate this consistency, those two probabilities and the point to compute them have to be determined. A Gaussian distribution is assumed inside the cell:

$$p(\mathbf{x}_m|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)) = \frac{1}{\sqrt{(2\pi)^3 |\boldsymbol{\Sigma}_i|}} \exp\left(-\frac{1}{2}(\mathbf{x}_m - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_m - \boldsymbol{\mu}_i)\right). \tag{3.38}$$

For the probability $p(\mathbf{x}_m|\mathbf{z}_t)$, it is assumed that the distance between $\mathbf{x}_m$ and $\mathbf{z}_t$ can be approximated by a Gaussian distribution centred on $\mathbf{z}_t$, with a variance σ_s^2 coming from the sensor [21]:

$$p(\mathbf{x}_m|\mathbf{z}_t) = \frac{1}{\sqrt{(2\pi)\sigma_s^2}} \exp\left(-\frac{\|\mathbf{x}_m - \mathbf{z}_t\|^2}{2\sigma_s^2}\right). \tag{3.39}$$

It is now necessary to find which point on the ray to consider for evaluating those two probabilities. NDT OM [21] proposes to evaluate them at the point $\mathbf{x}_M$ that maximises $p(\mathbf{x}_m|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ (remember that $\mathbf{x}_m$ is constrained to be on the ray). This point can be found analytically as shown by Saarinen et al. [21], the details of the minimisation are added here. Assuming with no loss of generality that the position $\mathbf{l^p}_t$ is the origin of the sensor, a point $\mathbf{x}_m$ on the ray can be expressed as:

$$\mathbf{x}_m = \lambda \mathbf{d} + \mathbf{l^p}_t, \tag{3.40}$$

with

$$\mathbf{d} = \mathbf{z}_t - \mathbf{l^p}_t. \tag{3.41}$$

Then, inserting equation (3.40) inside $p(\mathbf{x}_m|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ yields:

$$\begin{aligned}
\mathbf{x}_M &= \max_{\lambda} \left(\exp\left(-\frac{1}{2} (\lambda\mathbf{d} + \mathbf{l^p}_t - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1} (\lambda\mathbf{d} + \mathbf{l^p}_t - \boldsymbol{\mu}_i) \right) \right), \\
&= \min_{\lambda} \left((\lambda\mathbf{d} + \mathbf{l^p}_t - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1} (\lambda\mathbf{d} + \mathbf{l^p}_t - \boldsymbol{\mu}_i) \right), \\
&= \min_{\lambda} \left(\lambda^2 \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} \mathbf{d} + \lambda \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{l^p}_t - \boldsymbol{\mu}_i) + (\mathbf{l^p}_t - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1} \lambda\mathbf{d} \right), \\
&= \min_{\lambda} \left(\lambda^2 \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} \mathbf{d} + 2\lambda \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{l^p}_t - \boldsymbol{\mu}_i) \right).
\end{aligned} \tag{3.42}$$

At this point, the minimum is shown to exist (since $\mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} \mathbf{d}$ is strictly positive). If $h(\lambda) = \lambda^2 \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} \mathbf{d} + 2\lambda \mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{l^p}_t - \boldsymbol{\mu}_i)$, then the minimum λ^* is found by negating the derivative:

$$\begin{aligned}
&\frac{dh}{d\lambda}(\lambda^*) = 0, \\
\implies &\lambda^* = -\frac{\mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{l^p}_t - \boldsymbol{\mu}_i)}{\mathbf{d}^t \boldsymbol{\Sigma}_i^{-1} \mathbf{d}}.
\end{aligned} \tag{3.43}$$

Inserting λ^* into equation (3.40) gives $\mathbf{x}_M$.

Qualitatively, there is consistency between the ray and the distribution inside the cell if $p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ (equation (3.38), the point is well explained by the Gaussian distribution in the cell) is high and $p(\mathbf{x}_M|\mathbf{z}_t)$ (equation (3.39), the end point is close to $\mathbf{x}_M$) is high as well. On the other hand, if $p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ is low and $p(\mathbf{x}_M|\mathbf{z}_t)$ is low, this is consistent too: the ray passes far enough from the distribution. The inconsistency arises when $p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i))$ is high, which means a point should be observed, but $p(\mathbf{x}_M|\mathbf{z}_t)$ is

low, so $\mathbf{x}_M$ is still far from the end point. In this case, there is probably a false positive: the distribution models an obstacle that is absent or smaller. To summarise, the occupancy should be penalised more when $p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i))(1-p(\mathbf{x}_M|\mathbf{z}_t))$ is high.

For this reason, the occupancy update of a cell is defined as follows [21]:

$$p(\mathcal{C}_i|\mathbf{z}_t,\mathbf{l}_t) = \begin{cases} \alpha < 0.5 & \text{if the cell is empty,} \\ \beta > 0.5 & \text{if } \mathbf{z}_t \text{ is in the cell,} \\ 0.5 - \gamma p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i))(1-p(\mathbf{x}_M|\mathbf{z}_t)) & \text{otherwise.} \end{cases} \tag{3.44}$$

If the cell is empty or contains $\mathbf{z}_t$ (thus is considered occupied), the update is the same as standard occupancy mapping. If the cell contains a distribution, but not $\mathbf{z}_t$, then the log odd occupancy will be lowered by an amount that depends on the consistency between the ray and the distribution inside the cell. The more consistent they are, the closer the occupancy will be to 0.5.

The tuning constants α and β are positive in the interval $[0,1]$ and dictate the speed at which the cell occupancy changes. It has been noted by Saarinen et al. [21] that α should depend on the actual trajectory of the ray. If a ray can pass through a partially occupied cell, then the fact that it passes through a cell with no distribution does not mean that the cell is empty. It only shows that the part covered by the ray is empty. Thus, the choice of a constant value for α is a commodity due to the absence of a simple way to encode this information [21]. The tuning constant γ encodes the speed at which a partially occupied cell is penalised. It can be noted that since $p(\mathbf{x}_M|\mathcal{N}(\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i))(1-p(\mathbf{x}_M|\mathbf{z}_t)) \in [0,1]$, γ should be in $[0,0.5]$ to guarantee an occupancy between 0 and 0.5.

3.2.6.1 A raytracing example

The raytracing process is illustrated in Figure 3.2. On this figure, a single ray from the LIDAR to the measurement $\mathbf{z}_t$ is represented. This ray goes through four cells. The Gaussian distributions inside each cell are represented by red ellipses. There are four different

situation illustrated. In the cell $\mathcal{C}_1$, the ray crosses an occupied cell but is consistent with the distribution inside the cell. In the cell $\mathcal{C}_2$ the ray crosses an occupied cell as well, but this time is not consistent with the local distribution. The cell $\mathcal{C}_3$ is empty and the cell $\mathcal{C}_4$ contains the final point.

The first cell, $\mathcal{C}_1$, is not empty: it contains a distribution. Let $\boldsymbol{\mu}_1$ be the mean of this distribution, and $\boldsymbol{\Sigma}_1$ its covariance. The objective is to determine if the path of the ray is consistent with the Gaussian distribution inside the cell. In order to do this, the point $\mathbf{x}_M^1$ is computed for this cell by using Equations (3.40) to (3.43). This point represents the point of the ray that is the most likely to be generated from the distribution. Then, the actual probability that $\mathbf{x}_M^1$ is generated from the distribution, $p\left(\mathbf{x}_M^1|\mathcal{N}(\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)\right)$, is computed from equation (3.38). The probability that $\mathbf{x}_M^1$ can be explained by $\mathbf{z}_t$, $p\left(\mathbf{x}_M^1|\mathbf{z}_t\right)$ is also computed from equation (3.39). Looking at $\mathcal{C}_1$, both those probabilities are low: $\mathbf{x}_M^1$ is far enough of the distribution so that $p\left(\mathbf{x}_M^1|\mathcal{N}(\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)\right)$ is low, and far enough from $\mathbf{z}_t$ so that $p\left(\mathbf{x}_M^1|\mathbf{z}_t\right)$ is low as well. Using equation (3.44) to compute the cell occupancy update, the discussed case is the third one: the cell is not empty and $\mathbf{z}_t$ is not in the cell. $p(\mathcal{C}_1|z_t,l_t)$ is computed using $p\left(\mathbf{x}_M^1|\mathcal{N}(\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1)\right)$ and $p\left(\mathbf{x}_M^1|\mathbf{z}_t\right)$. Since those two probabilities are low, $p(\mathcal{C}_1|z_t,l_t)$ is close to 0.5. The log odd is applied to $p(\mathcal{C}_1|z_t,l_t)$ as shown in equation (3.34). In this case, the result is a small negative value. Finally, this value is added to the log odd occupancy of the cell $\mathcal{C}_1$, as per equation (3.36), while clamping the value using equation (3.37). Since in this case the ray is consistent with the distribution, the log odd occupancy of the cell was only slightly reduced.

The second cell, $\mathcal{C}_2$, is not empty as well. The exact the same steps as for $\mathcal{C}_1$ are applied. The difference is in the probability $p\left(\mathbf{x}_M^2|\mathcal{N}(\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_2)\right)$. This time, $\mathbf{x}_M^2$ can be well explained by the distribution, so $p\left(\mathbf{x}_M^2|\mathcal{N}(\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_2)\right)$ is close to one. Note that $p\left(\mathbf{x}_M^2|\mathbf{z}_t\right)$ is still low because $\mathbf{x}_M^2$ is still far from $\mathbf{z}_t$. Thus, the product of $p\left(\mathbf{x}_M^2|\mathcal{N}(\boldsymbol{\mu}_2,\boldsymbol{\Sigma}_2)\right)$ by $\left(1-p\left(\mathbf{x}_M^2|\mathbf{z}_t\right)\right)$ is close to one. Equation (3.44) then produces a probability $p(\mathcal{C}_2|z_t,l_t)$ that is low, which means that the value of the log odd occupancy will be very negative. The log odd occupancy of the cell will as a result be reduced by an important amount.

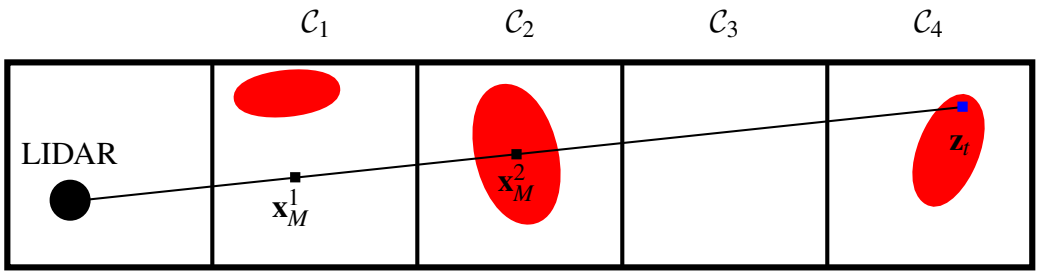


Figure 3.2: Illustration of the raytracing process. This process changes the occupancy inside each cell. Invalidated Gaussian distributions are deleted.

The cell $\mathcal{C}_3$ is empty, the occupancy update value is then α, as indicated in equation (3.44). Since $\alpha < 0.5$, the log odd occupancy of the cell is reduced.

Finally, the cell $\mathcal{C}_4$ is occupied, but contains the measurement $\mathbf{z}_t$. This is the second case of equation (3.44). The occupancy update is β, which is greater than 0.5, thus the log odd occupancy of the cell increases. The only case where the log odd occupancy of a cell increases is when this cell is observed as occupied.

It is important to note that the raytracing process only modifies the occupancy value inside a cell. The mean, covariance and number of points are modified during the cell fusion described in Section 3.2.4.

3.2.6.2 The complete NDT OM algorithm

It is now possible to combine the original NDT OM algorithm [21] with the registration [60] to create a map from a point cloud. Such a combination was proposed in 2013 by Stoyanov et al. [5]. A complete description of the procedure is proposed in Algorithm 3.2.

Several modifications can be brought to this algorithm. It is possible to use another way of registration, although in this case, the advantage is that a scan is not kept after use. Regarding the ray tracing process, which requires a lot of computational power, it is possible to reduce the number of rays while keeping a good precision [6]. Instead of using all the points p in $\mathcal{P}^{\text{global}}$, $\text{NDT}^*_{\text{global}}$ can be built beforehand and raytracing only uses the mean of the cell containing the point. To achieve this, the update equation (3.36) needs to

Algorithm 3.2: A complete mapping algorithm using NDT OM [5]

Input : A point cloud $\mathcal{P}$ in the sensor reference frame and a current NDT representation of the map $\text{NDT}_{\text{global}}$

Output: An updated NDT representation $\text{NDT}_{\text{global}}$ of the global map

Construct a local NDT representation $\text{NDT}_{\text{local}}$ from $\mathcal{P}$ (see Section 3.2.3);
Perform registration between $\text{NDT}_{\text{global}}$ and $\text{NDT}_{\text{local}}$ to get the rigid transformation $\mathbf{T}^{l\rightarrow g}$ between both (see Section 3.2.3);
Use $\mathbf{T}^{l\rightarrow g}$ to align $\mathcal{P}$ in the global reference frame : let $\mathcal{P}^{\text{global}}$ be this new point cloud;
for *all points* $p \in \mathcal{P}^{\text{global}}$ **do**
    Get all the cells $\mathcal{C}_1, \cdots, \mathcal{C}_k$ in $\text{NDT}_{\text{global}}$ traversed by the ray from the sensor to p;
    for *each cell* $\mathcal{C}_{i,i\in 1:k}$ **do**
        Update the log odd occupancy of $\mathcal{C}_i$ with equation (3.36) and (3.44);
    end
end
Compute an NDT representation from $\mathcal{P}^{\text{global}}$. Let it be $\text{NDT}^{*}_{\text{global}}$ (see Section 3.2.3);
for *all cells* $\mathcal{C} \in \text{NDT}^{*}_{\text{global}}$ **do**
    Fuse $\mathcal{C}$ into $\text{NDT}_{\text{global}}$ using RSC or ARSC (see Section 3.2.4);
end

be modified to add the occupancy as many times as the number of points in the final cell. The resulting algorithm is described in Algorithm 3.3. It greatly reduces the number of raytracing operations. For this reason, this idea has been kept in the proposed algorithm. Let N_p be the number of points in a new point cloud, N_g the number of Gaussian distributions created from this point cloud and M_g the number of Gaussian distributions currently in the map. The time complexities of the algorithm are: $\mathcal{O}(N_p)$ for the creation of an NDT map from the point cloud, $\mathcal{O}(M_g + N_g)$ for the matching (note that the quadratic complexity $\mathcal{O}(M_g N_g)$ is avoided since only a fixed neighbourhood of a given Gaussian distribution is considered), $\mathcal{O}(N_p)$ for the raytracing process of Algorithm 3.2 (cut down to $\mathcal{O}(N_g)$ in Algorithm 3.3) and $\mathcal{O}(N_g)$ for the final map fusion.

Algorithm 3.3: A modified version of the NDT OM algorithm, including a speed optimisation [6]

Input : A point cloud $\mathcal{P}$ in the sensor reference frame and a current NDT representation of the map $\text{NDT}_{\text{global}}$

Output: An updated NDT representation $\text{NDT}_{\text{global}}$ of the global map

Construct a local NDT representation $\text{NDT}_{\text{local}}$ from $\mathcal{P}$ (see Section 3.2.3);

Perform registration between $\text{NDT}_{\text{global}}$ and $\text{NDT}_{\text{local}}$ to get the rigid transformation $\mathbf{T}^{l\rightarrow g}$ between both (see Section 3.2.3);

Use $\mathbf{T}^{l\rightarrow g}$ to align $\mathcal{P}$ in the global reference frame : let $\mathcal{P}^{\text{global}}$ be this new point cloud;

Compute an NDT representation from $\mathcal{P}^{\text{global}}$. Let it be $\text{NDT}^{*}_{\text{global}}$ (see Section 3.2.3);

for *all cells* $\mathcal{C}^{*} \in \text{NDT}^{*}_{\text{global}}$ **do**

  Get the number of points n and mean $\boldsymbol{\mu}$ of cell $\mathcal{C}^{*}$;

  Get all the cells $\mathcal{C}_1, \cdots, \mathcal{C}_k$ in $\text{NDT}_{\text{global}}$ traversed by the ray from the sensor to $\boldsymbol{\mu}$;

  for *each cell* $\mathcal{C}_{i, i\in 1:k}$ **do**

    Update the log odd occupancy of $\mathcal{C}_i$ with equation (3.36) modified as follows: $l_{\text{odd}}(\text{occ}_i(t)) = l_{\text{odd}}(\text{occ}_i(t-1)) + n l_{\text{odd}}(\mathcal{C}_i|\boldsymbol{\mu}, l_t)$;

  end

end

for *all cells* $\mathcal{C} \in \text{NDT}^{*}_{\text{global}}$ **do**

  Fuse $\mathcal{C}$ into $\text{NDT}_{\text{global}}$ using RSC or ARSC (see Section 3.2.4);

end

3.3 NDT RC: normal distribution transform occupancy map with recentring

A problem with the NDT OM algorithm is the stationary state of the map: the map is expressed in a fixed reference frame, thus the robot can potentially leave the map. In order to allow movement beyond the limits of the initial map, it is possible to divide the map in submaps that are saved on the disk and reloaded when needed [5]. This solution is interesting if keeping the whole map is a requirement of the application. In order to perform obstacle avoidance, a coherent global map is not required, so it is proposed to use a map that would move with the robot without keeping out of bounds obstacles. Such a method does not rely on disk storage and is more efficient in memory, since a smaller part of the map has to be stored. It can be argued that the global coherence of the map is then negatively affected, but testing proves that precise global maps are still computed (see Section 3.5). On the other hand, the problem of loop closure present in NDT OM fusion [5] does not occur. If the formerly visited areas are stored, when the robot goes back to the same area, the matching is more difficult if the odometry drifted. A moving map does not suffer from this issue.

The proposed mapping algorithm differs from the original NDT OM fusion algorithm [5] on two points:

- a single map that moves with the robot is used, without storing submaps on the disk,
- the raytracing algorithm has been modified.

The algorithm was implemented in C++. The following section describes the algorithms that performs the recentring of the map and the raytracing.

3.3.1 The recentring algorithm and its implementation

3.3.1.1 Principle of the algorithm

The principle of the recentring algorithm is the following: the map starts centred on the robot. When the robot goes further from the centre than a fixed threshold d_{thrc}, the map moves so that its new centre is the cell that the robot is currently occupying. Note that the map can only be translated this way, it does not rotate. This process is illustrated in Figure 3.3. The robot is initially at the centre of the grid map, populated by some Gaussians represented in blue and red. After a first movement, the distance from the centre of the map to the robot is still inferior to d_{thrc}, thus no recentring is needed. However, after the second movement, the robot is now further from the centre than d_{thrc}. The new cell at the centre of the map is the cell in which the robot currently is, outlined in magenta. The map is shifted so that this cell goes to the centre. All the cells that are outside the new map, such as the cell hosting the red Gaussian, are discarded.

It follows that the choice of d_{thrc} does not influence the mapping precision directly, only the speed of the algorithm. The smallest d_{thrc} is, the more often the map has to be recentred, which slows the algorithm down. However, if d_{thrc} is too large, then some points from a point cloud might appear out of bound while being close to the robot. This poses a safety issue. Thus, it is recommended to set d_{thrc} so that $(s_{map} - d_{thrc})/2$, with s_{map} the size of the map, is a reasonable security distance with regard to the speed of the robot.

The challenge to tackle when implementing the recentring is the aliasing issue: care has to be taken to not overwrite a cell before it is moved. The exact implementation is described in Algorithm 3.5.

3.3.1.2 Data structures used

The following considers a 2D map to allow for a clearer representation, but the extension to 3D is straightforward.

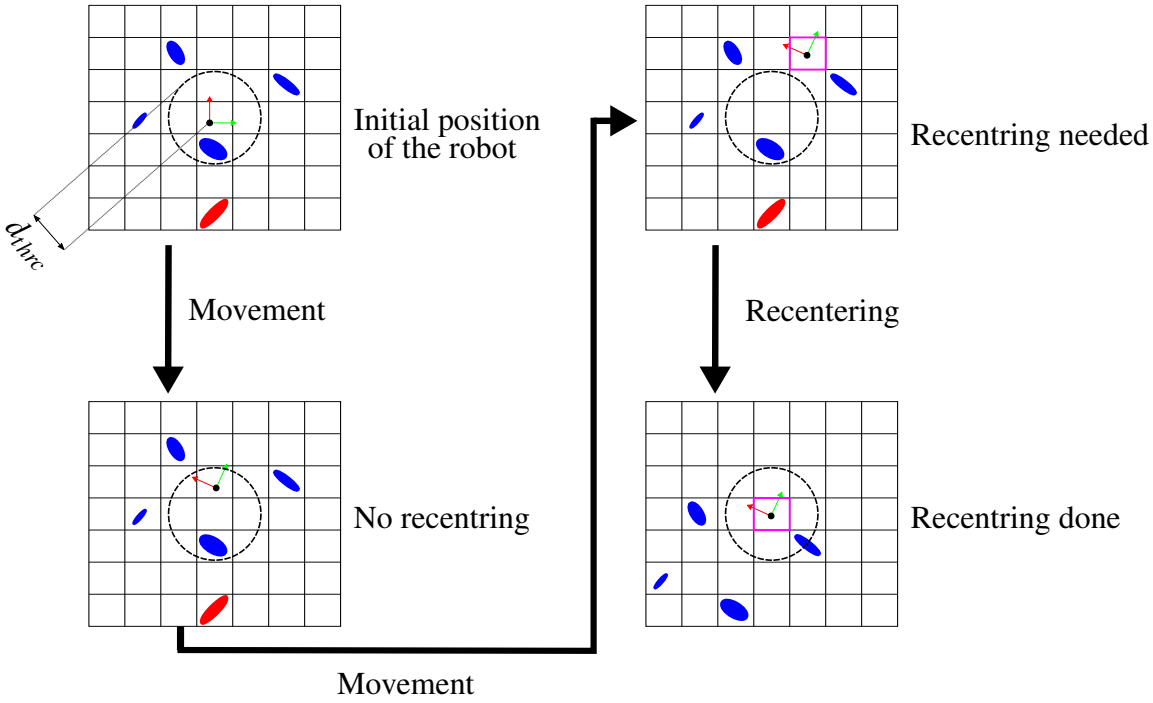


Figure 3.3: Illustration of the recentring algorithm

The chosen implementation is to use three different structures to describe the map. Those structures are described in Figure 3.4.

The first element is the map, which only hosts indices, but no actual data. Initially, the map is full of zeros. The indices in the map link each cell to a second array: the list of active cells. The list of active cells initially has a single component, which is special and will remain unused and empty. A cell is considered active if one of those two conditions is met:

- there is a 3D point to add to the cell,
- the raytracing algorithm passes by the cell.

Note that this way, an empty cell is considered active and thus tracked, since the occupancy value of an empty cell should be kept. Once a cell is active, it stays active until it goes out of the map limits due to recentring. The data of the cell (i.e. its mean, covariance, occupancy, and number of points, such as described in Section 3.2.3) are stored inside the list of active cells. In the example of Figure 3.4, there are 4 active cells: 1, 2, 4 and 5. The list of deleted cells hosts the indices in the list of active cells that correspond to cells

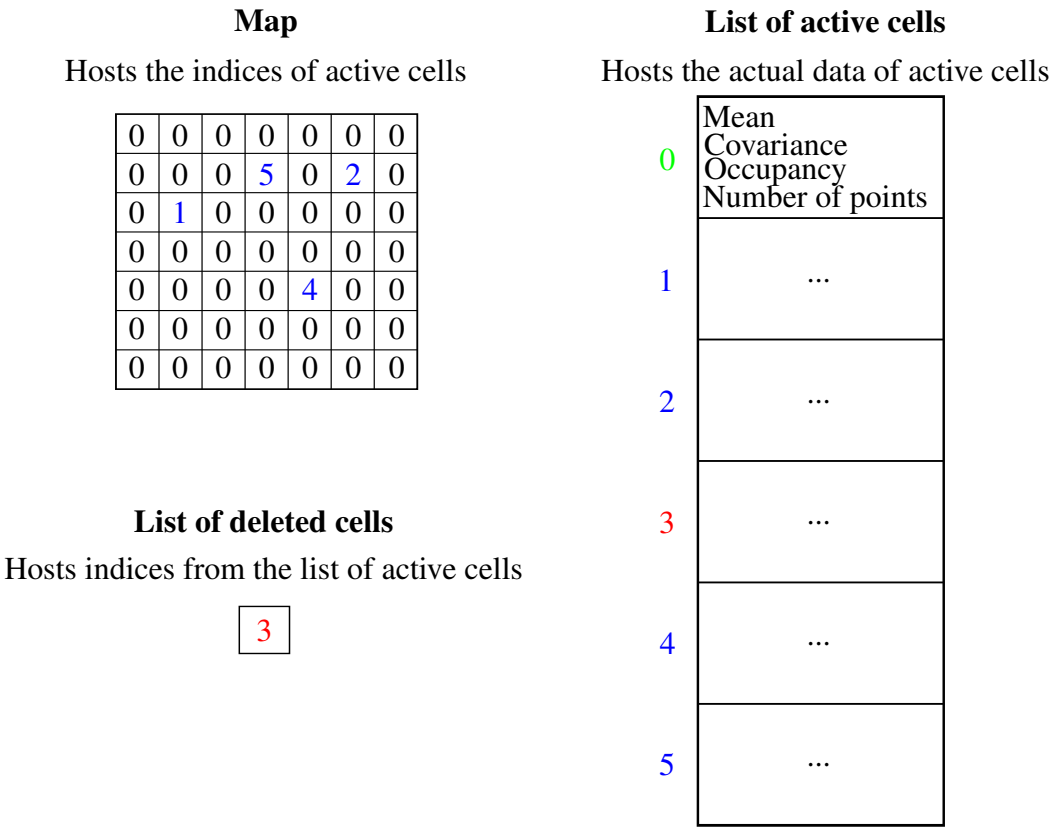


Figure 3.4: Structures used to manage the map

deleted after the recentring. In this case, that would be the cell 3, which is not present on the map anymore. When a new active cell is added, if the list of deleted cells is non empty, then its first element is chosen to host the new data of the active cell. If the list of deleted cells is empty, then an element is added to the list of active cells. This allows to keep a list of active cells that is relatively dense.

The list of active cells represents an overhead compared to a map that would only host the actual cell data. However, it has two advantages.

First, using a list of active cells results in a lower memory usage in the vast majority of situations. Indeed, let s_{double} be the size, in bits, of a double precision floating point number, and $s_{unsigned}$ the size of an unsigned integer. For n_{active} cells, $n_{deleted}$ cells, and a total number of cells in the map noted n_{total}, the memory requirement of our scheme is, in bits:

$$m_{list} = n_{total} s_{unsigned} + n_{active} \left(10 s_{double} + s_{unsigned}\right) + n_{deleted} s_{unsigned} \tag{3.45}$$

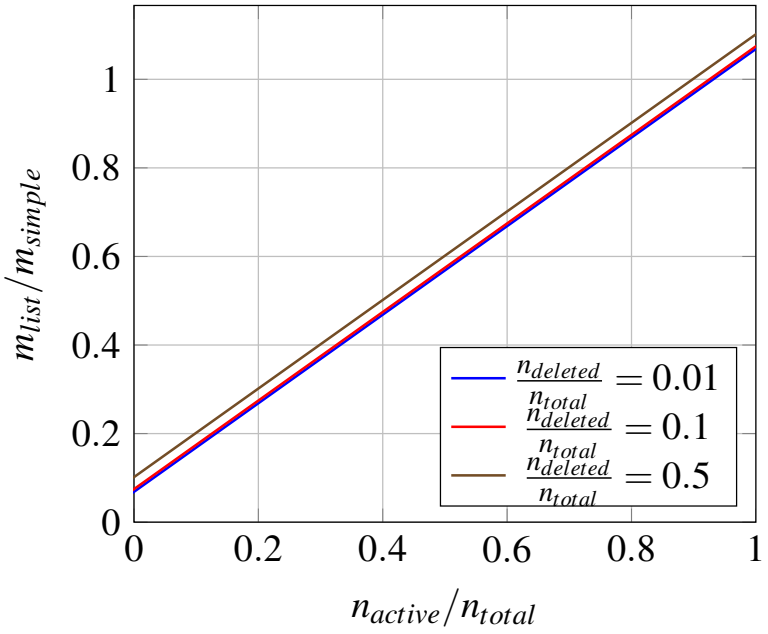


Figure 3.5: Ratio of the memory needed compared to a standard approach

On the other hand, the memory requirement of a direct map would be:

$$m_{simple} = n_{total}\left(10 s_{double} + s_{unsigned}\right) \tag{3.46}$$

For standard values of 64 bits for s_{double} and 32 bits for $s_{unsigned}$, the ratio m_{list}/m_{simple} is plotted in Figure 3.5 as a function of n_{active}/n_{total}, with different quantity of deleted cells in the list of active cells.

Figure 3.5 shows that until the map reaches more than 85 % of the cells occupied, which is unlikely is a standard environment, the proposed scheme is more efficient in memory.

The second advantage is speed: if the active cells are separated, it is faster to loop through them for common operations, rather than looping through all the cells of the map. The matching stage is a good example, where it is required to loop through all active cells in the NDT formed by a new scan.

3.3.1.3 The recentring algorithm

What can be noted from this implementation is that the link from the map to the list of active cells is direct, however, the opposite is not true. While looping through the active cells, it is not possible to get the corresponding map cell for the active cells which do not have a Gaussian inside because of the absence of mean. This is a problem for the recentring algorithm because, for speed reasons, it is desirable to loop through the active cells to move them. It is thus desirable to be able to find a map cell corresponding to an active cell. For this reason, if an active cell does not host a Gaussian, its mean is set to the centre of the cell. This ensures that each active cells has a valid mean, which allows to go back to its spatial location (i.e. its map index).

This allows to perform recentring in an efficient way: the idea is to loop through the list of active cells and use the mean to get back to the corresponding map cell. Then, there are two distinct possibilities:

- the index stored in the map cell is the actual index of the considered active cells. In this case, this index is moved at its new location in the map (note that there can be a new index here, in this case it is overwritten, see the second point). If the new location is outside the map, the index is instead added to the list of deleted cells. In both cases, the former index in the map is deleted.

- If the index stored in the map cell does not match the active cell, this implies that the index of this active cell was overwritten. In this case, the index of the active cell is still copied to the new location, but the initial value is not erased. Again, if the new location is outside the map, the index of the active cell is added to the list of deleted cells.

In both cases, the mean stored in an active cell is updated.

This process is illustrated in Figure 3.6 for a map displacement of $(-2,0)$. Figure 3.6a depicts the initial state. There is already one deleted cell, and the list of active cells contains, among others, the means to go back to the map coordinates. In order to simplify

the depiction, the map coordinates are directly represented in the list of actives cells. The active cell number 1 (Figure 3.6a), due to the displacement of the map, goes out of it. Since the cell of coordinates $(0,3)$ in the map hosts the index 1, this index can be erased. Because this index goes out of the map, it is added to the list of deleted cells. Note that a deleted cell is lost, if encountered again, it will be treated as a new cell. The result is depicted in Figure 3.6b.

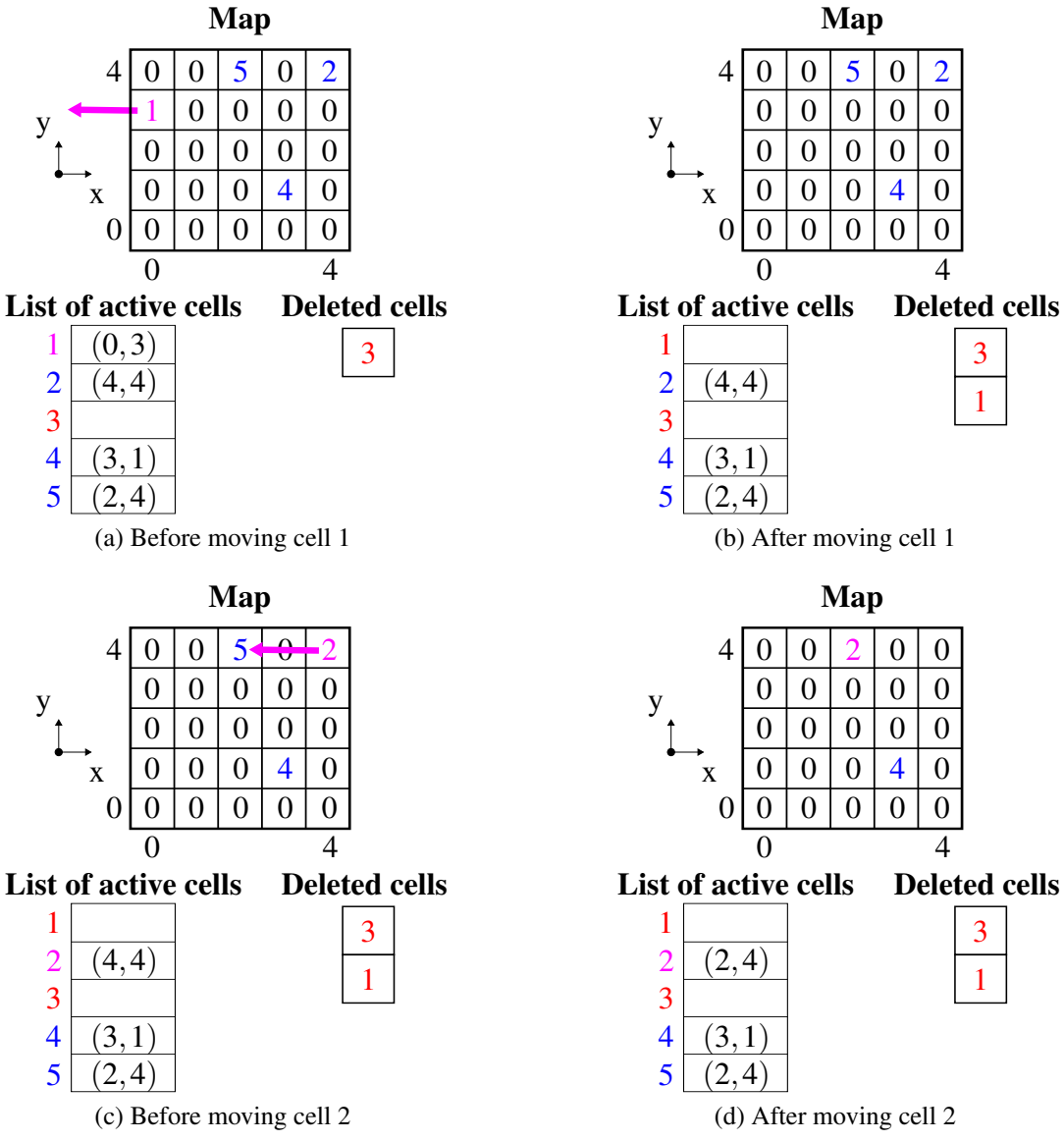


Figure 3.6: Illustration of the recentring algorithm for a considered displacement of $(-2,0)$ (part 1)

Now, the next cell in the list is cell number 2 (Figure 3.6c). Since the cell of coordi-

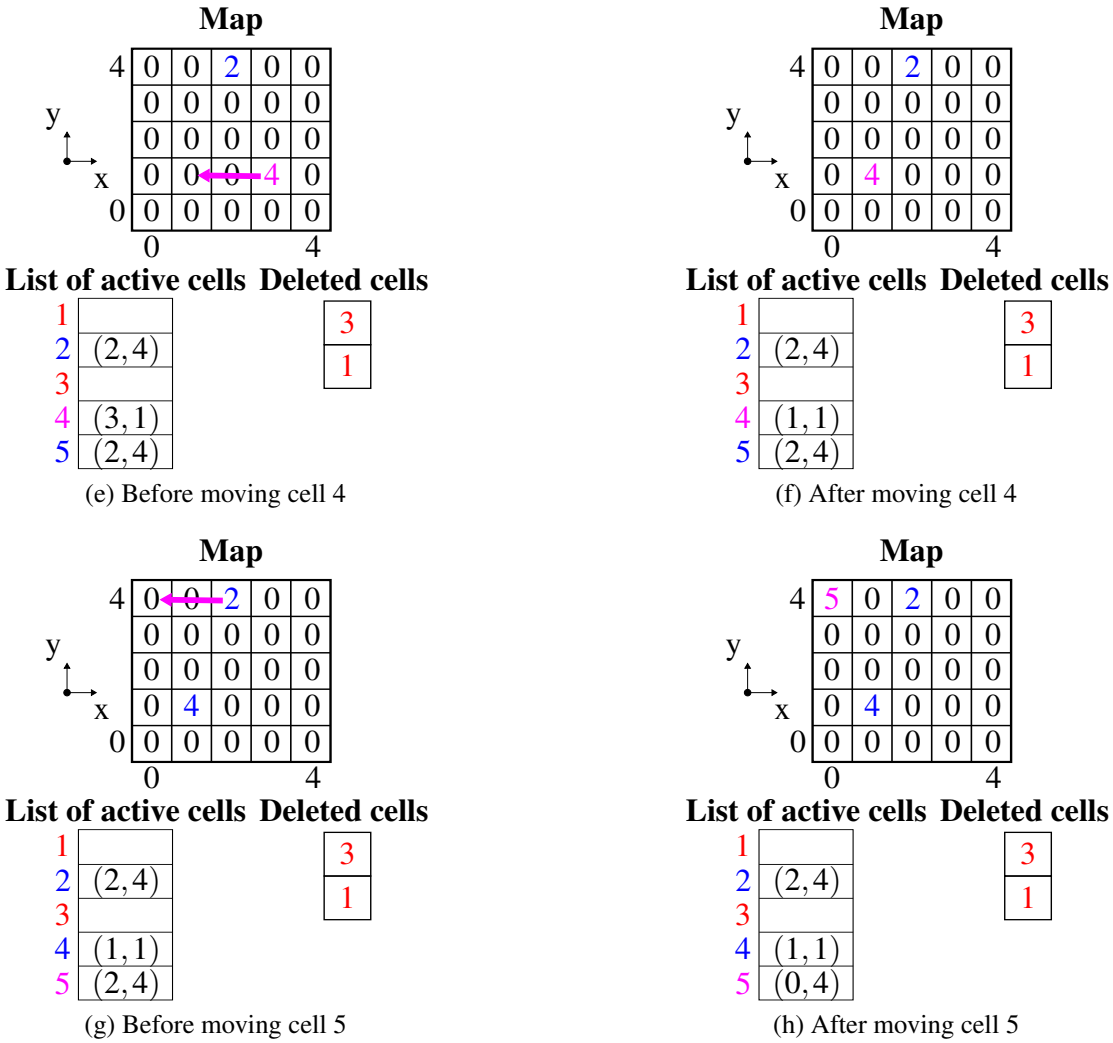


Figure 3.6: Illustration of the recentring algorithm for a considered displacement of $(-2,0)$ (part 2)

nates $(4,4)$ hosts the index 2, actual index of the current active cell, it can be erased and moved to its next destination, $(2,4)$ where it overwrites 5. The internal coordinates of cell 2 are modified as well (see Figure 3.6d).

The next active cell, 3 is skipped because the cell is deleted. The next current active cell is 4 (Figure 3.6e). Since the cell of coordinates $(3,1)$ hosts the index 4, the content of the map cell $(3,1)$ can be deleted and moved to $(1,1)$. The content of the active cell is adjusted (Figure 3.6f).

The remaining cell is the number 5 (Figure 3.6g). This time, the map coordinates of cell 5 are $(2,4)$, but at those coordinates, the index in the map is 2, and not 5. Thus, the

content of the map cell $(2,4)$ is *not* erased, but 5 is still placed into the map cell $(0,4)$, and the content of the active cell is updated accordingly (see Figure 3.6h).

3.3.1.4 Complete NDT RC algorithm

When a new measurement is added at time t, the recentring algorithm needs three rigid transforms: $\mathbf{T}_{t-1}^{m\to w}$, $\mathbf{T}_{t-1}^{s\to m}$ and $\mathbf{T}_{t-1\to t-2}^{s\to m}$. Initially, all those transform can be initialised as the identity.

The detailed algorithm, named NDT RC (standing for NDT ReCentring) in the following, is described in pseudo code by Algorithm 3.4, the recentring sub function is written separately in Algorithm 3.5 and illustrated in Figure 3.6. The list of active cells is abbreviated L_{ac}, the list of deleted cells L_{dc}, and the map M. The existing structures for the map are indexed by g (so L_{ac}^{g}, L_{dc}^{g} and M^{g} define the current map). Since the current time is t, the rigid transforms $\mathbf{T}_{t-1}^{m\to w}$, $\mathbf{T}_{t-1}^{s\to m}$ and $\mathbf{T}_{t-1\to t-2}^{s\to m}$ are defined.

Similarly to the rigid transform conventions described in Section 3.2.1, the point clouds are indexed according to the reference frame in which they are defined and their time. For instance, $\mathcal{P}_t^s$ refers to the point cloud obtained at time t and expressed in the sensor reference frame. The details about the update of the rigid transforms $\mathbf{T}_{t-1}^{s\to m}$ and $\mathbf{T}_{t-1\to t-2}^{s\to m}$ are as follows. Let $\tilde{\mathbf{T}}_{t\to t-1}^{s\to m}$ be an estimation of the movement of the robot between t and $t-1$ and $\tilde{\mathcal{P}}_t^m$ the resulting estimation of the LIDAR point cloud in the frame of reference of the map. $\tilde{\mathbf{T}}_{t\to t-1}^{s\to m}$ is initialised to the inter frame transform between $t-2$ and $t-1$, in other words:

$$\tilde{\mathbf{T}}_{t\to t-1}^{s\to m} = \mathbf{T}_{t-1\to t-2}^{s\to m}. \tag{3.47}$$

Then the resulting point cloud is estimated:

$$\begin{aligned}
\tilde{\mathcal{P}}_t^m &= \tilde{\mathbf{T}}_t^{s\to m}\mathcal{P}_t^s, \\
&= \mathbf{T}_{t-1}^{s\to m}\tilde{\mathbf{T}}_{t\to t-1}^{s\to m}\mathcal{P}_t^s, \\
&= \mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}\mathcal{P}_t^s.
\end{aligned} \tag{3.48}$$

Algorithm 3.4: The complete mapping algorithm: NDT RC

Input : A point cloud $\mathcal{P}_t^s$ in the sensor reference frame at time step t
Output: M^g, L_{ac}^g, L_{dc}^g are updated and $\mathbf{T}_t^{m\to w}$, $\mathbf{T}_t^{s\to m}$, $\mathbf{T}_{t\to t-1}^{s\to m}$ are created

// Transform the input cloud into the map reference frame. The initial guess is the last inter frame transform, and is applied here

$\tilde{\mathcal{P}}_t^m = \mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}\mathcal{P}_t^s$;

Build M^l and L_{ac}^l from $\tilde{\mathcal{P}}_t^m$ (see Section 3.2.3);

Register $\{M^l, L_{ac}^l\}$ with $\{M^g, L_{ac}^g\}$, initial guess is the identity, the output is $\mathbf{T}^\Delta$ (see Section 3.2.3) If the registration fails, $\mathbf{T}^\Delta$ is the identity;

// Update the position of the robot, as well as the next inter frame transform

$\mathbf{T}_t^{s\to m} = \mathbf{T}^\Delta\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}$;

$\mathbf{T}_{t\to t-1}^{s\to m} = \left(\mathbf{T}_{t-1}^{s\to m}\right)^{-1}\mathbf{T}^\Delta\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}$;

// Create the actual point cloud that will be added

$\mathcal{P}_t^m = \mathbf{T}_t^{s\to m}\mathcal{P}_t^s$;

// Get from it an NDT representing the new scan correctly aligned

Build M^l and L_{ac}^l from $\mathcal{P}_t^{\text{final}}$ (see Section 3.2.3);

// Fuse $\{M^l, L_{ac}^l\}$ with $\{M^g, L_{ac}^g\}$

// Start with the raytracing to update the occupancy

for *all cells* $\mathcal{C}^* \in L_{ac}^l$ **do**

  Get the number of points n and mean $\boldsymbol{\mu}$ of cell $\mathcal{C}^*$;

  // M^g is used to perform the raytracing

  // Get all the cells $\mathcal{C}_1, \cdots, \mathcal{C}_k$ in L_{ac}^g traversed by the ray from the sensor to $\boldsymbol{\mu}$

  $\mathcal{C}_1, \cdots, \mathcal{C}_k = raytrace\left(\text{trans}\left(\mathbf{T}_t^{s\to m}\right), \boldsymbol{\mu}\right)$(see Algorithm 3.6);

  for *each cell* $\mathcal{C}_{i,i\in 1:k}$ **do**

    Update the log odd occupancy of $\mathcal{C}_i$ with equation (3.36) modified as follows: $l_{\text{odd}}\left(\text{occ}_i\left(t\right)\right) = l_{\text{odd}}\left(\text{occ}_i\left(t-1\right)\right) + nl_{\text{odd}}\left(\mathcal{C}_i|\boldsymbol{\mu}, l_t\right)$;

  end

end

// Then fuse the cells themselves

for *all cells* $\mathcal{C} \in L_{ac}^l$ **do**

  Fuse $\mathcal{C}$ into L_{ac}^g using RSC or ARSC (see Section 3.2.4);

end

// Performs recentring if needed

$recentre\_map\left(\mathbf{T}_t^{s\to m}, L_{ac}^g, M^g\right)$;

Algorithm 3.5: The recentring algorithm : *recentre_map*

Input : $\mathbf{T}_t^{s\to m}$, L_{ac}^g, M^g
Output: Updated $\mathbf{T}_t^{s\to m}$, L_{ac}^g, L_{dc}^g, M^g, created $\mathbf{T}_t^{m\to w}$

if $norm\left(\text{trans}\left(\mathbf{T}_t^{s\to m}\right)\right) > d_{thrc}$ **then**
  Compute the integer offset vector (in cells) necessary to recentre the map $\mathbf{t}_{\text{offset}}^{\text{cells}}$;
  // This is the offset in meters, m_{res} is the map resolution
  $\mathbf{t}_{\text{offset}}^{\text{meters}} = m_{res}\mathbf{t}_{\text{offset}}^{\text{cells}}$;
  // Update the rigid transforms
  $\text{trans}\left(\mathbf{T}_t^{s\to m}\right) = \text{trans}\left(\mathbf{T}_t^{s\to m}\right) - \mathbf{t}_{\text{offset}}^{\text{meters}}$;
  $\text{trans}\left(\mathbf{T}_t^{m\to w}\right) = \text{trans}\left(\mathbf{T}_{t-1}^{m\to w}\right) - \mathbf{t}_{\text{offset}}^{\text{meters}}$;
  // Apply the recentring to the cells
  for *all* $i \in [\![1, size\left(L_{ac}^g\right)]\!]$ **do**
    $\mathcal{C} = L_{ac}^g(i)$;
    // As noted above, for each cell in L_{ac}, the physical centre can be computed
    $\mathbf{x}_{\text{centre}}^m = get\_cell\_centre(\mathcal{C})$;
    $\mathbf{x}_{\text{new\_centre}}^m = \mathbf{x}_{\text{centre}}^m - \mathbf{t}_{\text{offset}}^{\text{cells}}$;
    if $\mathbf{x}_{\text{new\_centre}}^m$ *is still inside* M^g *limits* **then**
      // If the cell, once translated, is still inside the map, the destination map index is populated
      $M^g\left(\mathbf{x}_{\text{new\_centre}}^m\right) = i$;
      // Then it is decided whether to erase the index in the former map cell
      if $M^g\left(\mathbf{x}_{\text{centre}}^m\right) == i$ **then**
        $M^g\left(\mathbf{x}_{\text{centre}}^m\right) = 0$;
      // We then update the mean of the cell
      Get mean $\boldsymbol{\mu}$ of cell $\mathcal{C}$;
      $\boldsymbol{\mu} = \boldsymbol{\mu} - \mathbf{t}_{\text{offset}}^{\text{meters}}$
    else
      // In this case, the new cell is not inside the limits of the map: the cell needs deletion. As above, the need to erase the former map index is checked
      if $M^g\left(\mathbf{x}_{\text{centre}}^m\right) == i$ **then**
        $M^g\left(\mathbf{x}_{\text{centre}}^m\right) = 0$;
      // In both cases, add the index to L_{dc}^g and delete $\mathcal{C}$
      $delete(\mathcal{C})$;
      $L_{dc}^g \leftarrow i$
    end
  end
else
  $\mathbf{T}_t^{m\to w} = \mathbf{T}_{t-1}^{m\to w}$;
end

This point cloud is used to build the first temporary NDT. The registration between this NDT and the global NDT obtained from the current map gives $\mathbf{T}^{\Delta}$ so that:

$$\begin{aligned}\mathcal{P}_t^m &= \mathbf{T}^{\Delta}\tilde{\mathcal{P}}_t^m, \\ &= \mathbf{T}^{\Delta}\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}\mathcal{P}_t^s.\end{aligned} \tag{3.49}$$

This equation allows us to get the updated pose of the sensor:

$$\mathbf{T}_t^{s\to m} = \mathbf{T}^{\Delta}\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}. \tag{3.50}$$

The inter frame transform can then be updated as well:

$$\begin{aligned}\mathbf{T}_{t\to t-1}^{s\to m} &= \left(\mathbf{T}_{t-1}^{s\to m}\right)^{-1}\mathbf{T}_t^{s\to m}, \\ &= \left(\mathbf{T}_{t-1}^{s\to m}\right)^{-1}\mathbf{T}^{\Delta}\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}.\end{aligned} \tag{3.51}$$

At this point, $\mathbf{T}_t^{s\to m}$ and $\mathbf{T}_{t\to t-1}^{s\to m}$ have been updated and can be used to compute $\mathcal{P}_t^m$. A new NDT is then created from $\mathcal{P}_t^m$ and fused with the current map, as explained in Algorithm 3.4.

3.4 Raytracing

This section describes our implementation of the raytracing process to update the cells. Let the ray starts at point $\mathbf{x}_{\text{start}}$ and ends at point $\mathbf{x}_{\text{end}}$. The goal is to compute all the cells traversed by the ray between $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{end}}$ to then modify their occupancy as specified in Section 3.2.6.

In order to get a cell on the path of the ray, a point inside this cell has to be found. The very first cell is always the location of the robot. To find the next cell, the principle of the algorithm is to detect the point on the intersection between a ray and the first cell face. Since this point is on the limit between two or more cells, a small fixed distance is added to this point in the direction of the ray, giving a second point which is now properly inside the next cell. The process is then repeated until reaching the final cell for this ray.

First, the normalised direction $\mathbf{d}_r$ of the ray is computed as:

$$\mathbf{d}_r = \frac{\mathbf{x}_{\text{end}} - \mathbf{x}_{\text{start}}}{\|\mathbf{x}_{\text{end}} - \mathbf{x}_{\text{start}}\|}. \tag{3.52}$$

Then, the function f_{dir} is applied to each component of $\mathbf{d}_r$ to create $\mathbf{d}_{rq}$, which is the quantized direction:

$$f_{\text{dir}}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0. \end{cases} \tag{3.53}$$

Let $\mathbf{c}$ be the centre of the current cell, then if r is the resolution of the map, the boundary vector $\mathbf{b}$ can be defined as:

$$\mathbf{b} = \mathbf{c} + \frac{r}{2}\mathbf{d}_{rq} \tag{3.54}$$

$\mathbf{b}$ contains the coordinates of each of the planes that can be traversed by the ray.

$\mathbf{b}$ represents the common vertex of the faces of the cell that can potentially be traversed by the ray in the $\mathbf{d}_r$ direction. This means that one of the three coordinates of $\mathbf{b}$ will represent the closest intersection, i.e. the intersection with the face of the current cell. The other coordinates are then determined by the equation of the ray. A point on the ray

can be defined as:

$$\mathbf{x}_r = \mathbf{x}_{\text{start}} + \lambda \mathbf{d}_r. \tag{3.55}$$

Thus, in order to get the intersection between the ray and any of the cell border, equation (3.56) is solved for λ.

$$\begin{aligned} \mathbf{b} = &\quad \mathbf{x}_{\text{start}} + \lambda \mathbf{d}_r, \\ \Rightarrow \quad \lambda_i = &\quad \frac{\mathbf{b}(i) - \mathbf{x}_{\text{start}}(i)}{\mathbf{d}_r(i)} \quad \text{if } \mathbf{d}_r(i) \neq 0, \text{for } i \in [\![1,3]\!]. \end{aligned} \tag{3.56}$$

Since at least one component of $\mathbf{d}_r$ has to be non zero, equation (3.56) provides at least one valid value for λ, at most three. The smallest of those values is chosen, let it be λ_m.

Finally, a small distance constant ε_{rt} is chosen, and a point in the next cell, $\mathbf{x}_{\text{next}}$ is computed as:

$$\mathbf{x}_{\text{next}} = \mathbf{x}_{\text{start}} + (\lambda_m + \varepsilon_{\text{rt}})\, \mathbf{d}_r \tag{3.57}$$

The point $\mathbf{x}_{\text{next}}$ is then used to find the cell containing it. From there, $\mathbf{x}_{\text{start}}$ is updated as $\mathbf{x}_{\text{next}}$, and the algorithm continues until reaching the cell containing $\mathbf{x}_{\text{end}}$. The complete algorithm is written in Algorithm 3.6.

A note about the implementation: in practice, the list of traversed cells is not returned, but updated directly in the raytracing method.

The proposed raytracing algorithm is compared with the NDT OM implementation available from the GitHub repository of the authors[2].

Figure 3.7 and Figure 3.8 presents a visual comparison: the proposed algorithm is on the left, the NDT OM implementation on the right. The start and end points are the same for both algorithms. The cell size is 20 cm in this simulation. The number of cells

[2] `https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic`, last pull done on Tuesday 27th November, 2018

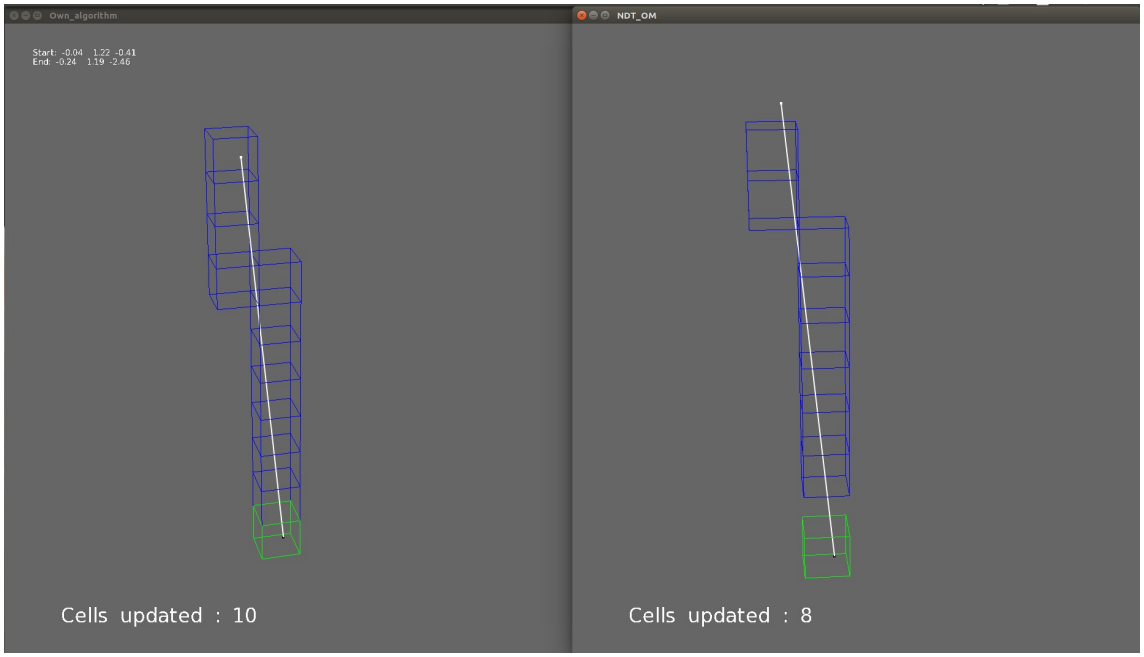
Algorithm 3.6: The raytracing algorithm : *raytracing*

Input : Start and end points $\mathbf{x}_{\text{start}}$, $\mathbf{x}_{\text{end}}$
Output: A list L_t of the traversed cells

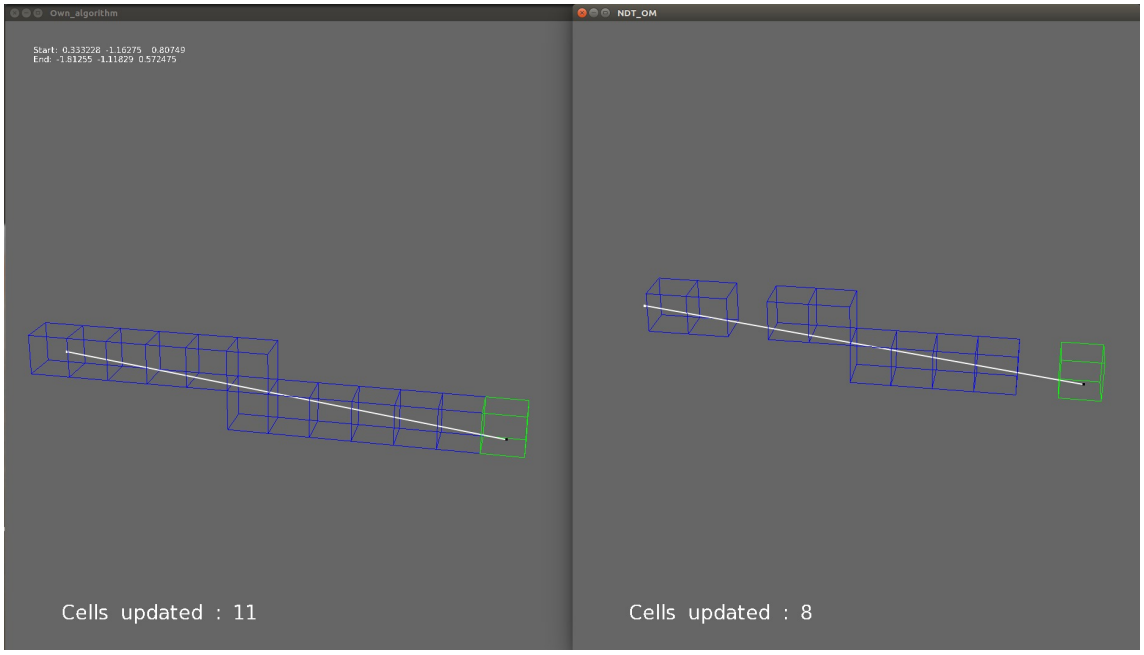
$L_t = \emptyset$;
$\mathbf{x}_{\text{current}} = \mathbf{x}_{\text{start}}$;
$\mathbf{d}_r = \frac{\mathbf{x}_{\text{end}} - \mathbf{x}_{\text{start}}}{\|\mathbf{x}_{\text{end}} - \mathbf{x}_{\text{start}}\|}$;
$\mathbf{d}_{rq} = f_{\text{dir}}(\mathbf{d}_r)$;
gap $= \|\mathbf{x}_{\text{end}} - \mathbf{x}_{\text{current}}\|^2$;
while gap > 0 **do**
  Get cell $\mathcal{C}$ containing $\mathbf{x}_{\text{current}}$;
  $L_t \leftarrow \mathcal{C}$;
  if $\mathcal{C}$ *contains* $\mathbf{x}_{\text{end}}$ **then**
    Returns L_t;
  end
  Get centre $\mathbf{c}$ of $\mathcal{C}$ and its size r;
  $\mathbf{b} = \mathbf{c} + \frac{r}{2}\mathbf{d}_{rq}$;
  `// The maximum value of a double is noted` $d_{\max}$
  $\lambda = [d_{\max}; d_{\max}; d_{\max}]$;
  for $i \in [\![1,3]\!]$ **do**
    if $\mathbf{d}_r(i) \neq 0$ **then**
      $\lambda(i) = \frac{\mathbf{b}(i) - \mathbf{x}_{\text{current}}(i)}{\mathbf{d}_r(i)}$;
    end
  end
  $\lambda_m = \min(\lambda(1), \lambda(2), \lambda(3))$;
  $\mathbf{x}_{\text{next}} = \mathbf{x}_{\text{current}} + (\lambda_m + \varepsilon_{\text{rt}})\mathbf{d}_r$;
  gap $= \|\|\mathbf{x}_{\text{current}} - \mathbf{x}_{\text{end}}\|\| - \|\|\mathbf{x}_{\text{next}} - \mathbf{x}_{\text{end}}\|\|$;
  $\mathbf{x}_{\text{current}} = \mathbf{x}_{\text{next}}$;
end

traversed, excluding the start and end cells, is shown on each image. The start and end points are shown as well. When the cells are mostly traversed from one face to the opposite (i.e. there is no "diagonal"), the algorithms act similarly (see Figure 3.7). However, when the number of "diagonal" crossing increases, the proposed implementation registers more cells, as shown on Figure 3.8 from the number of updated cells. This means that the proposed implementation is more sensitive to cells that the ray only slightly goes through. Experiments (see Section 3.5.1) have shown that such a change would require an adaptation of parameters α and β of the occupancy update process.

In order to quantify the difference in number of cell traversed by both algorithms, ran-

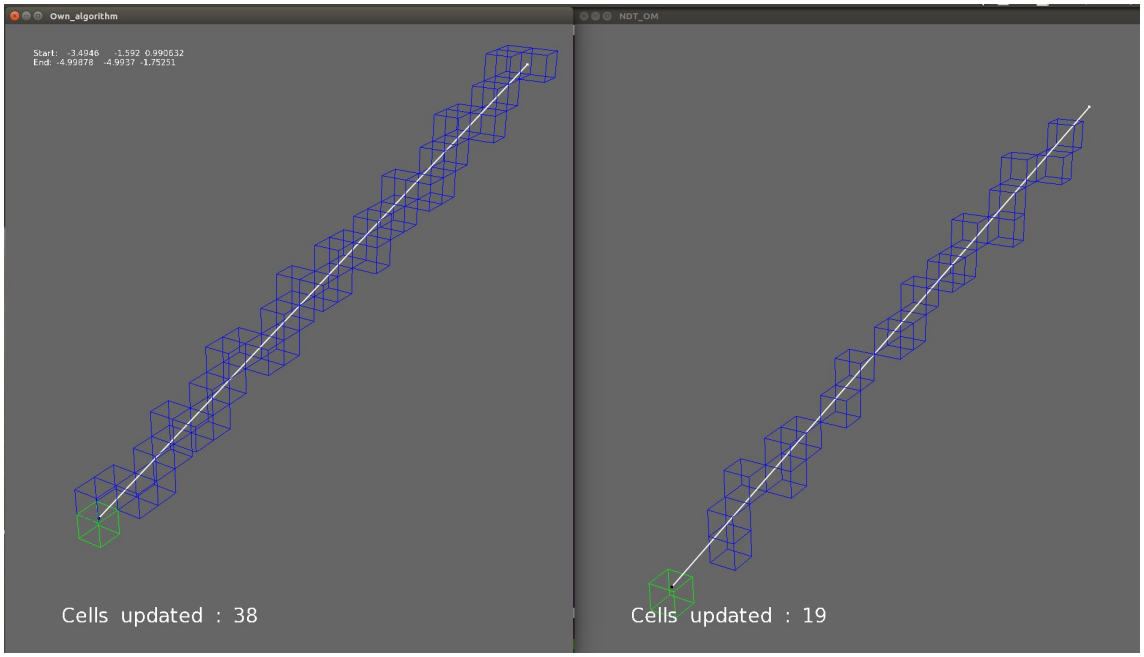


(a)

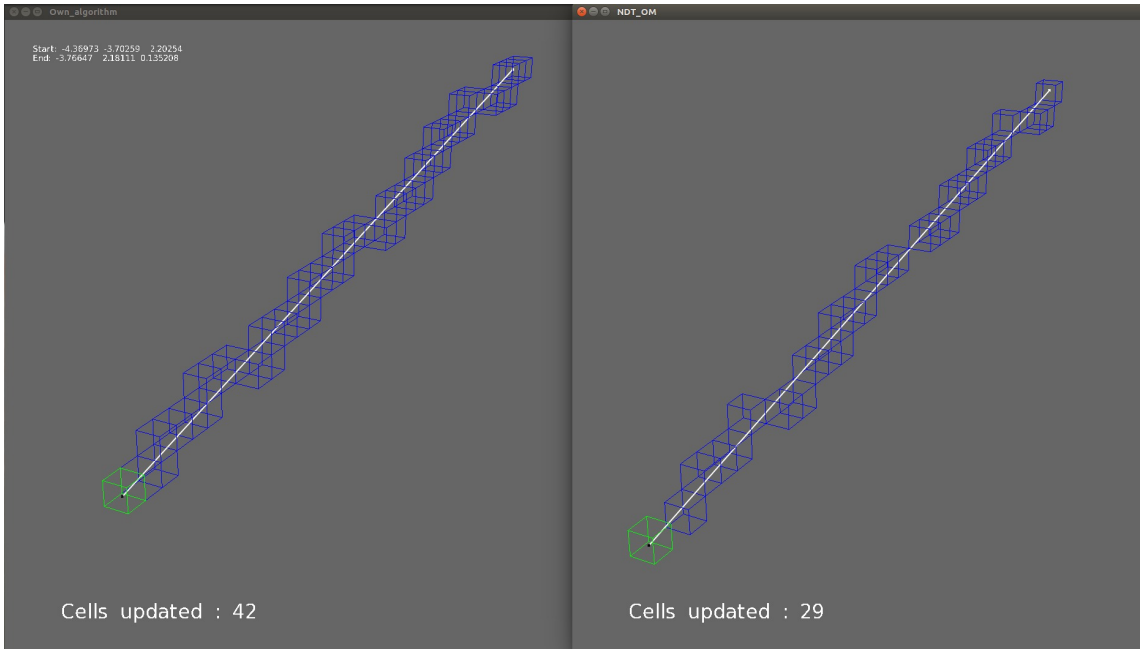


(b)

Figure 3.7: Comparison of raytracing methods. The proposed algorithm on the left, NDT OM on the right



(a)



(b)

Figure 3.8: Comparison of raytracing methods on more complicated examples. The proposed algorithm on the left, NDT OM on the right

Figure 3.9: Comparison of the number of traversed cells in function of the length of a ray for both methods. Points are randomly generated

dom start and end points are generated. The random number generator follows a uniform distribution between $-5\,\mathrm{m}$ and $5\,\mathrm{m}$. The cell size is still $20\,\mathrm{cm}$. 1×10^6 pairs of points were randomly generated, then the result was subsampled to allow printing. A scatter plot of the result is shown in Figure 3.9, along with a linear fit. Note that the linear fit was performed on the full data.

Figure 3.9 shows that the proposed method goes through approximately 69 % more cells than the NDT OM implementation on average. Moreover, there is a physical lower bound for the number of cell needed to go through a given distance. Let l be the distance between the end and start of the ray. Let r be the size of a cubic cell. If the first and last cell are not counted, the minimum number of cells needed to cover the distance l is N_{lim}:

$$N_{lim}(l) = \max\left(0, \frac{l}{\sqrt{3r^2}} - 2\right). \tag{3.58}$$

This lower physical limit is plotted in orange in Figure 3.9. While the proposed method never goes below this physical limit, the NDT OM implementation has some points below it, which indicates missed cells in the algorithm.

3.5 Testing

In this section, the proposed algorithm is compared with the standard NDT algorithm on several datasets.

3.5.1 Tuning of the raytracing with the TUM RGB-D SLAM dataset

In order to test the cell fusion scheme, the authors of the NDT OM algorithm [21] (section 4.1) used a down sampled version of a publicly available dataset from the TUM RGB-D

SLAM repository [83]. This dataset is small, it contains 47 point clouds taken from a Kinect RGB-D camera in an office. In order to test the quality of the map, it is possible to accumulate all the scans into a single point cloud (using the odometry ground truth), construct an NDT from this scan, and compare it to an NDT obtained by the fusion of each NDT separately (while still using the odometry ground truth). It is proposed to perform the same operation to tune the raytracing.

The evaluation uses several metrics to compare the two resulting NDTs:

- The mean error between the mean of two overlapping cells $\boldsymbol{\mu}_{\text{error}}$.
- The mean likelihood L between two overlapping cells over all cells:

$$L = \exp\left(\frac{-(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}{2}\right). \tag{3.59}$$

  The closer this value is to one, the better the map matches the ground truth.

- The number of false positives (Fpos) and false negatives (Fneg) (i.e. cells that are considered occupied and should not be, and cells that are considered empty and should not be). The number of cells that are occupied both in the ground truth and the evaluation NDT are mentioned as the valid cells. The mean error and mean likelihood are evaluated over this number of cells. When the matrix inversion in equation (3.59) fails, this is counted as an error.

The NDT OM implementation used for this test was from the GitHub repository of the authors[3]. A resolution of 0.2 m is used, the matching search range for cell neighbours to compute the objective function is set to 2. The ARSC parameter M_{max} is set to 5000. The occupancy limit is set to 255. The values α, β, γ of equation (3.44) are set to their default value of 0.45, 0.6 and 0.1 respectively. The operating system used is Ubuntu 16.04.

The results are reported in Table 3.1 for different values of α and β.

[3] `https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic`, last pull done on Tuesday 27th November, 2018

Table 3.1: Error metrics of the TUM RGB-D SLAM dataset

Name	α	β	$\boldsymbol{\mu}_{\text{error}}$	L	Fpos	Fneg	Valid cells	Errors
NDT OM	0.45	0.6	1.18 mm	0.996	0	217	1821	139
NDT RC	0.45	0.6	1.25 mm	0.996	6	400	1653	125
NDT RC	0.39	0.8	1.18 mm	0.996	6	274	1775	129
NDT RC	0.39	0.9	1.25 mm	0.996	6	213	1826	129
NDT RC	0.45	0.8	1.18 mm	0.996	6	233	1815	130
NDT RC	0.45	0.9	1.11 mm	0.996	6	183	1864	131

For reference, $\boldsymbol{\mu}_{\text{error}}$ and L as reported in the original NDT OM paper [21] for those parameters are 1.5 mm and 0.995 respectively, so the current implementation seems to perform a little bit better. This difference might be explained by the treatment of false positives and false negatives, which is not explained in the original paper.

Regarding the results, it is seen that the mean error and mean likelihood do not change much (compared to the baseline, ± 0.07 mm or 5.9 % for the mean error, the mean likelihood stays the same). However, the number of false negatives can vary by as much as 84 %. This difference can be explained by the differences in the raytracing algorithm: since the proposed raytracing implementation passes through more cells, the occupancy in the cells that are partly occupied will decrease as well. This is why an increase in β will decrease the false negatives. However, care has to be taken when doing this: increasing β means more confidence in the scan points' location. Thus, increasing β can be complemented by decreasing α, in order to increase the negative occupancy propagation from raytracing.

3.5.2 Evaluation of the mapping algorithm

Evaluating the mapping quality is difficult, since a precise ground truth is required. That is complicated to get in real world scenarios, especially in dynamic environments. For this reason, an indirect approach is adopted: since the mapping and the odometry are linked, the quality of the odometry can be considered as an indication of the quality of the mapping. The odometry is easier to evaluate, since there are a number of public datasets

available [84, 85]. For this reason, the NDT RC algorithm was tested on the quality of the odometry it produces.

In order to evaluate the proposed algorithm, two publicly available datasets are used: the Ford Campus Vision and Lidar Data Set [85] (referred to as the Ford dataset in the following), and the KITTI odometry dataset [84] (referred to as the KITTI dataset). The Ford dataset was chosen because the basis of the NDT RC algorithm, that is the NDT OM [21], was evaluated on it with a fusion mechanism to allow the robot to have a greater movement range [5]. It is then interesting to be able to compare to it. The KITTI dataset [84] has been chosen because it is a widely used tool: at the time of writing, the online benchmark[4] contains 88 entries.

Both datasets are acquired outdoor using a car. The Ford dataset was acquired in urban conditions, while the KITTI dataset contains 11 sequences acquired in various conditions. Regarding the sensor used, both datasets use a Velodyne HDL-64E LIDAR[5], at 10 Hz, to acquire the data. The range of this LIDAR is 120 m. The Ford dataset uses an Applanix POS-LV 420 INS with Trimble GPS[6] as a combination of GPS and IMU to measure the ground truth. The KITTI dataset uses an OXTS RT 3003[7] localisation system to provide the ground truth.

3.5.2.1 Default parameters and runtime computation

Unless indicated otherwise, the parameters used for the tests are presented in Table 3.2 (see Section 3.2 and 3.3 for what those parameters refer to). Note that the results reported on both the KITTI and Ford datasets are obtained with those same parameters to avoid over fitting. The matching initialisation was made using the last valid inter frame transform, i.e. at time t, the matching is initialised with $\mathbf{T}^{s\to m}_{t-1\to t-2}$.

The runtime shown is the wall clock time, not the CPU time, which means that it should give a realist representation of the speed. By default, the time reported includes

[4] `http://www.cvlibs.net/datasets/kitti/eval_odometry.php`
[5] `https://velodynelidar.com/hdl-64e.html`
[6] Datasheet available at `https://www.applanix.com/products/poslv.htm`
[7] `https://www.oxts.com/products/rt3000/`

Table 3.2: Values of the parameters used in the proposed algorithm

Parameter	**Value**
Cell size	2.2 m
α	0.45
β	0.9
γ	0.1
Map width and depth	250 m
Map height	40 m
d_{thrc}	10 m
d_1	1
d_2	0.05
Neighbourhood size for matching	2
M_{max}	500
ε_{rt}	1 mm
K_{occ}	255

the whole algorithm, as described by Algorithm 3.4. Note that this program is entirely single threaded, there is no multithreading involved in any of the parts, and it runs entirely on the CPU. The computer used to run the tests has an Intel Core i7-6700 CPU, with a base frequency of 3.4 GHz and 16 GB of RAM.

3.5.3 Evaluation on the Ford dataset

3.5.3.1 Error metrics

The main error metric on this dataset is the absolute trajectory error [83] (ATE). The ATE is obtained after alignment of both trajectories [83], and measures the absolute error in translation. The script used for the alignment can be found online[8]. With the conventions defined in Section 3.2.1, if $\mathbf{T}_i^{gt}$ represents the rigid transform describing the position of the vehicle at time i, and $\mathbf{T}_i$ the estimated rigid transform at the same time, the error is

[8] `https://vision.in.tum.de/data/datasets/rgbd-dataset/tools`

defined as: $e_i = \|\text{trans}\left(\mathbf{T}_i^{gt} - \mathbf{T}_i\right)\|$. The ATE can then be computed as a mean, standard deviation, RMSE, median, min, or max over this set, i.e. for the mean and RMSE:

$$\begin{aligned} \text{ATE}_{\text{MEAN}}\left(e_{1:n}\right) &= \frac{1}{n}\sum_{i=1}^{n} e_i, \\ \text{ATE}_{\text{RMSE}}\left(e_{1:n}\right) &= \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}. \end{aligned} \tag{3.60}$$

In the following, if no indication is given, the metric used is the mean ATE.

3.5.3.2 Results on the Ford dataset

Note that in order to get the full data, the Ford dataset was regenerated by using the Matlab code provided with the dataset. First, the influence of the five principal parameters is investigated, all evaluated against the resolution of the map. Those parameters are: the values of α, β, γ, which govern the occupancy update during raytracing, the maximum occupancy per cell, K_{occ} and the limit number of points M_{max} inside a cell.

Figure 3.10 shows the variation of the ATE and runtime with the value of α and β. α is an indication of the confidence that the cell is empty when a ray goes through it, while β quantifies the probability that a cell is occupied when a point is detected inside it.

In order to keep the figure readable, errors of more that 9 m are omitted from the graph, they are listed here with the model $(\text{Resolution}, \alpha, \beta)$: $(0.6, 0.39, 0.8)$, $(0.6, 0.45, 0.9)$, $(0.8, 0.39, 0.7)$, $(1.2, 0.39, 0.7)$, $(1.6, 0.45, 0.8, 38555)$, $(2.2, 0.39, 0.7)$, $(2.6, 0.39, 0.7)$, $(2.8, 0.45, 0.8)$, $(3, 0.45, 0.7)$, $(3, 0.39, 0.8)$, $(3, 0.45, 0.8)$, $(3.2, 0.39, 0.8)$, $(3.4, 0.39, 0.9)$, $(3.2, 0.45, 0.9)$. Those outliers are indicative of tracking failure, they happen more often at low or high cell sizes.

In order for the matching to work reliably, the cell size should be large enough to capture significant features of the environment. However, if the cells are too large, the Gaussian approximation for an obstacle inside the cell might not hold anymore, resulting in matching errors. It is thus expected for the error to diminish up to the optimal cell size,

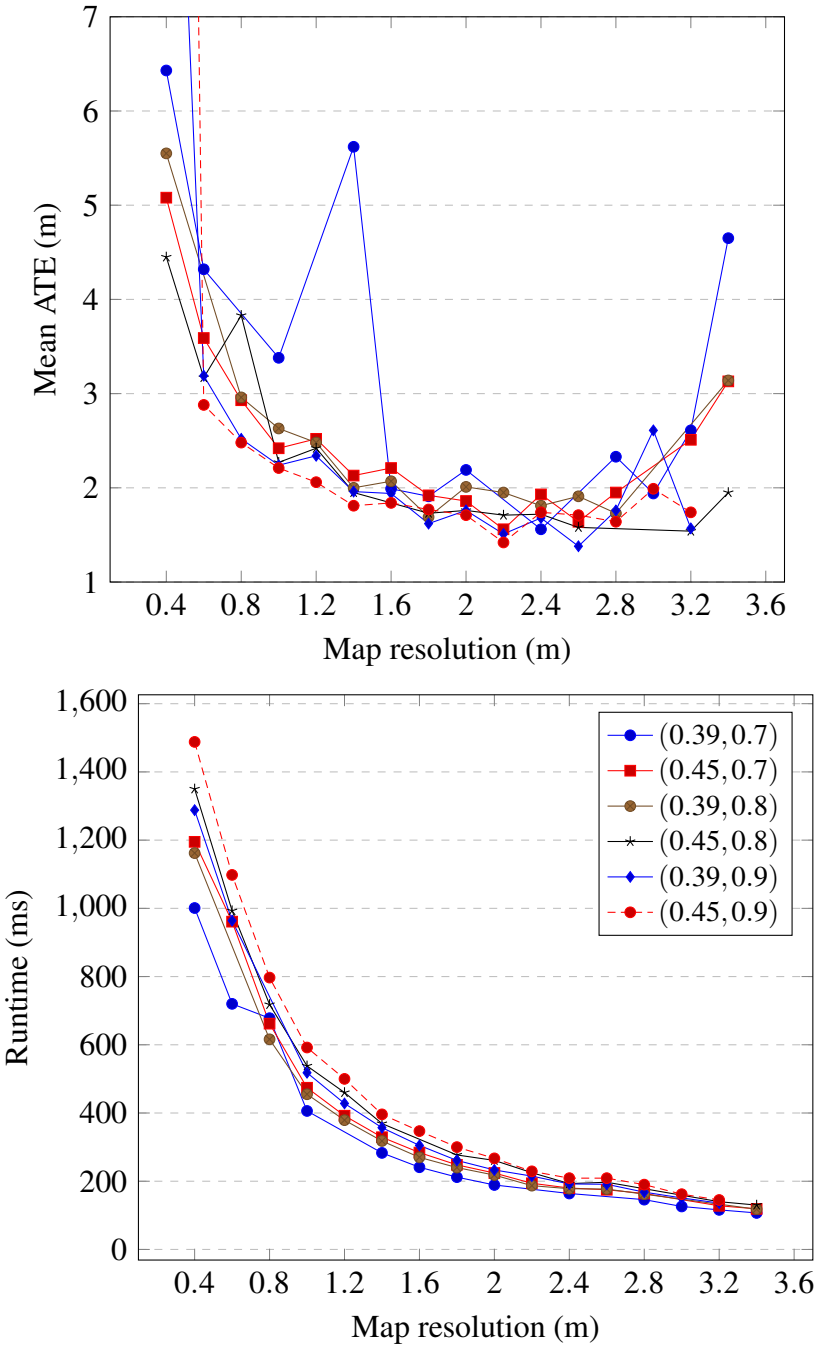


Figure 3.10: Evolution of the mean ATE and runtime depending on parameters α and β. The legend indicates a pair (α, β)

then increase when the cell size is too large to represent the environment accurately. The results show that increasing the cell size results in a fast reduction of the error up to around 1.4 m, with the error starting to increase at a cell size around 2.4 m. The mean ATE is the lowest around a resolution of 2.2 m. A good combination of stability and performance seems to be a couple (α, β) of $(0.45, 0.9)$, which gives a mean ATE of 1.42 m for a resolution of 2.2 m. The runtime is shown to be a function of the map resolution, although the values of α and β have an influence as well. The lower α is, the more punitive the raytracing is. Thus a lower α will tend to suppress more cells. On the other hand, a higher β will tend to create more cells. It is reasonable to consider that the runtime is a function of the number of active cells. Thus, theoretically, a low α and low β should have the lowest runtime, while a high α and high β should have the highest. This is precisely what is observed in Figure 3.10.

Figure 3.11 illustrates the influence of the parameter γ, which governs how punitive the raytracing is when a ray passes through a cell that is occupied by a Gaussian. There is no visible trend between between the values 0.1 and 0.3 for γ, except that a lower value of γ seems more stable: when $\gamma = 0.3$, the algorithm failed at resolution 1.2 m and 2.2 m. A higher value of γ means that cells hosting Gaussians are penalised by a larger loss of occupancy when traversed by a ray. Thus in theory, a lower runtime for higher values of γ is expected. This is however not the case, which indicates that the difference in number of invalidated cells hosting Gaussians is low between those the value of those two parameters.

Next, the influence of the parameter M_{max} is examined, which governs the sliding average when points are added into a cell: a higher M_{max} indicates a map slower to adapt to dynamic changes. The result is shown in Figure 3.12. The difference made by M_{max} is shown to be resolution dependent, with small values of M_{max} preferable at higher cell sizes. It can be hypothesised that smaller values of M_{max} would perform better in a dynamic environment such as the KITTI, however this is not observed at lower cell sizes. This parameter should have no influence on the runtime, which is demonstrated in this

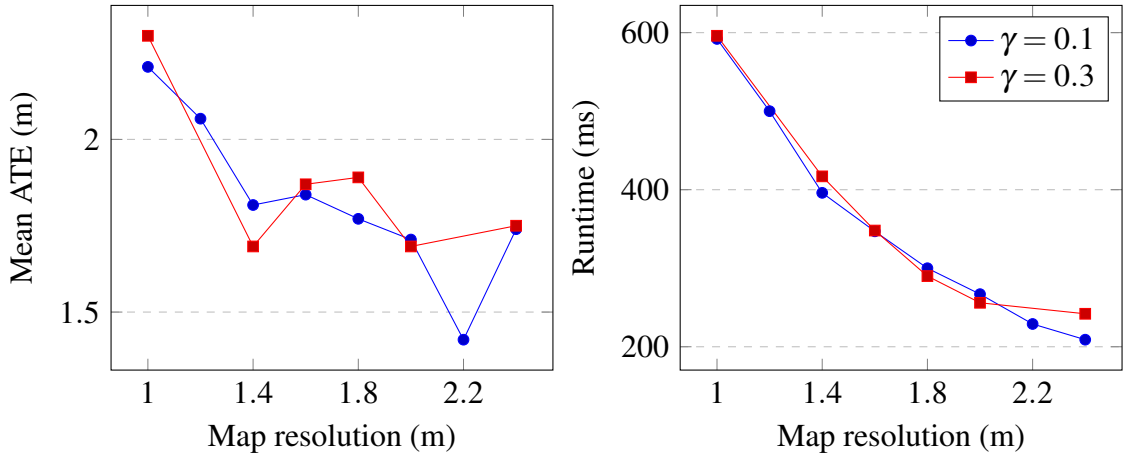


Figure 3.11: Evolution of the mean ATE and runtime depending on the γ parameter

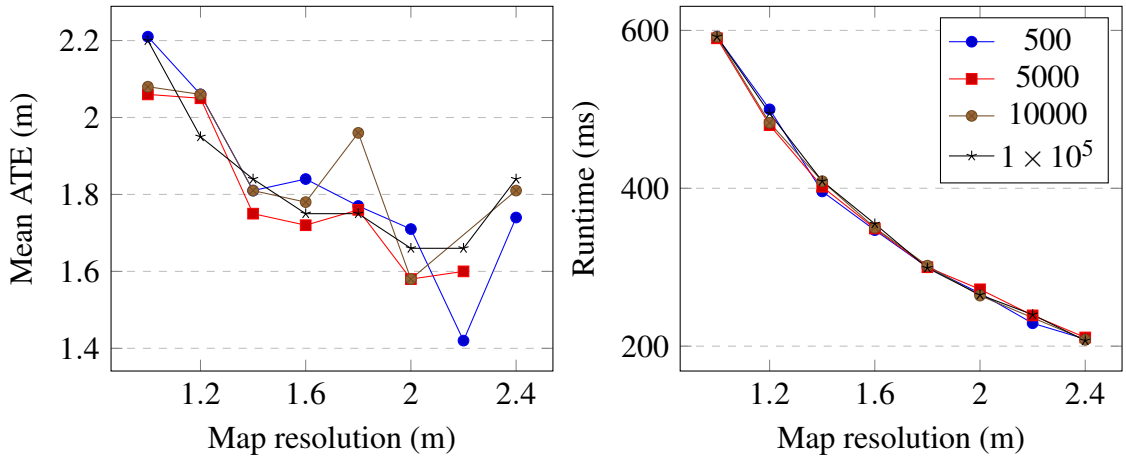


Figure 3.12: Evolution of the mean ATE and runtime depending on parameter M_{max}. The legend shows different values of M_{max}

figure.

The maximum occupancy threshold K_{occ} governs how fast cells are created eliminated in case of dynamic changes. The influence of this parameters is shown in Figure 3.13. The influence of the error is resolution dependent, with no clear trend. The higher value tends to show more stability across changes in resolution. It is difficult to forecast the influence of K_{occ} on the runtime since decreasing this parameter means that cells are both created and deleted more often. Given the results, it looks like this effect balances itself since no significant changes are observed for different values of K_{occ}.

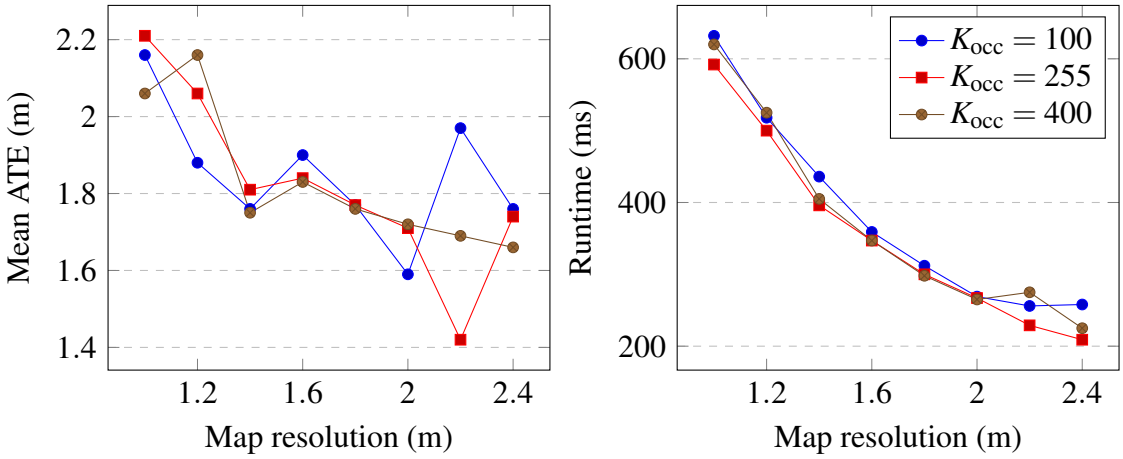


Figure 3.13: Evolution of the mean ATE and runtime depending on parameter K_{occ}

Next, the influence of the size of the map on the mean ATE is investigated. The LIDAR used has a range of 120 m, so the default size, 250 m, should be large enough to host the whole scan. Figure 3.14 examines the loss of precision that occurs when the whole LIDAR scan cannot fit into the map anymore. Note that the method fails for a map size of 140 m at the resolution 1.2 m and for the map size of 200 m at the resolution 1.0 m. Those outliers are not including in the graph to keep it readable. A map size of 140 m leads to a larger error compared to bigger sizes. The difference between 190 m, 220 m and 250 m depends on the resolution, with the lowest error achieved by the 250 m map which performs slightly better as the cell size increases. The runtime is barely affected by the size of the map in this dataset, meaning that the number of occupied cells at the edge of the map is small since their disappearance does not lead to a significant reduction in runtime.

Next, the influence of the raytracing algorithm is examined. Since the efficiency of the raytracing method is linked to the values of α and β, the NDT OM implementation of raytracing is tested with several values of β (the default value of β proposed by NDT OM is 0.6) and α fixed to 0.45. NDT RC uses a value of β of 0.9. The comparison is shown in Figure 3.15. Errors larger than 5 m are not shown. This happened for the NDT OM implementation for the following combination $(\text{resolution}, \beta)$ of value: $(1, 0.6)$, $(1, 0.9)$,

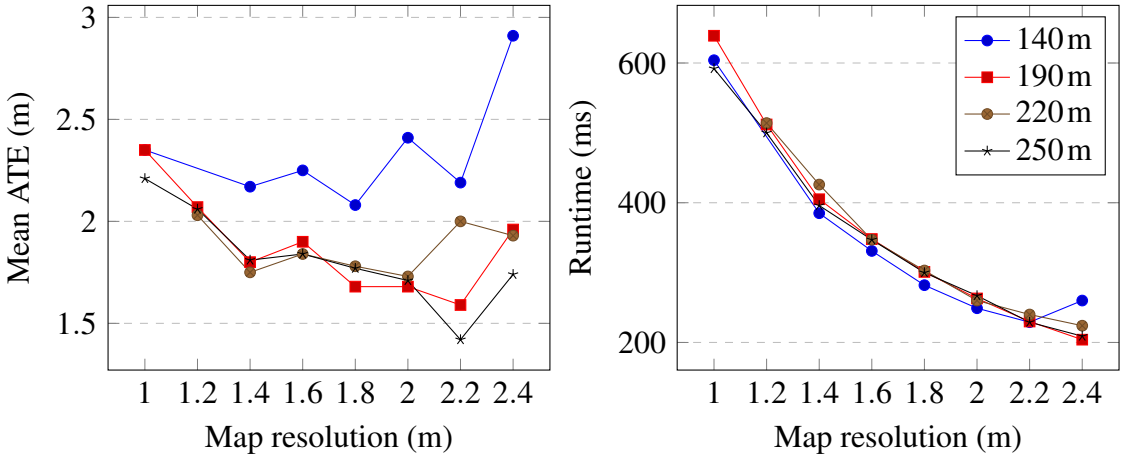


Figure 3.14: Evolution of the mean ATE and runtime depending on the size of the map. The legend shows different map sizes (the size is considered as width and depth)

$(1.2, 0.6)$ and $(1.6, 0.7)$. This test shows that the proposed raytracing implementation is more reliable and achieves the lowest overall error. NDT RC runs faster than the the NDT OM implementation of raytracing with the comparable value of $\beta = 0.9$.

In order to compare the results visually, a plot of the trajectory achieved by NDT RC using parameters specified in Table 3.2 compared to the ground truth is shown in Figure 3.16.

In order to facilitate future comparisons, some additional error statistics are reported in Table 3.3, using the parameters specified in Table 3.2: the mean of the ATE (the current metric), the root mean squared ATE, the median of the ATE, the standard deviation of the ATE, the minimum and maximum of the ATE. The end of trajectory error produced by NDT RC is 1.87 m with those parameters.

In Table 3.4, the average runtime is detailed. The reported runtime durations are: the time needed to import the point cloud and create an NDT model out of it, the time necessary to match this NDT model with the existing map, the time needed to update the existing map with this new scan, and the time to execute the recentring algorithm. Although those times are only for the most accurate result, the time distribution percentage among the different parts of the algorithm is overall similar. The matching represents more than 95 % of the execution time. The time needed by the recentring algorithm is

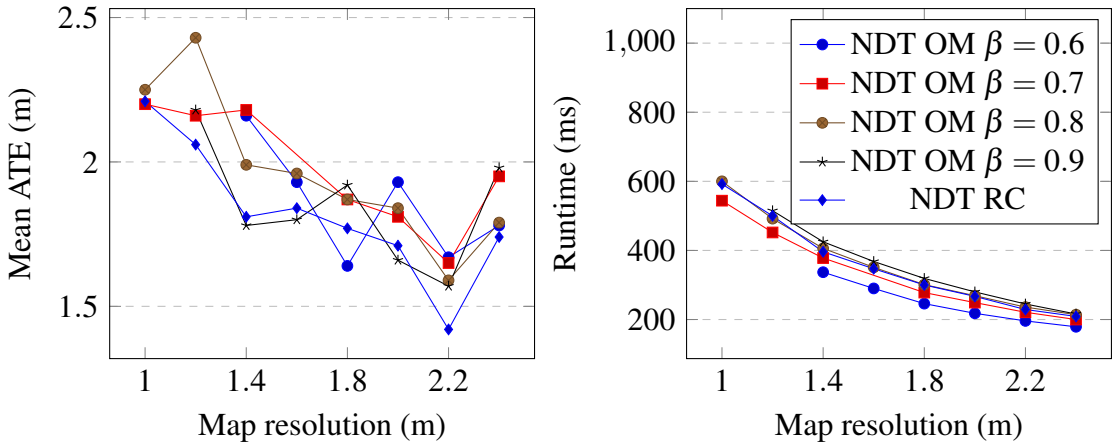


Figure 3.15: Comparison of the mean ATE and runtime between the proposed raytracing method and the NDT OM implementation of raytracing

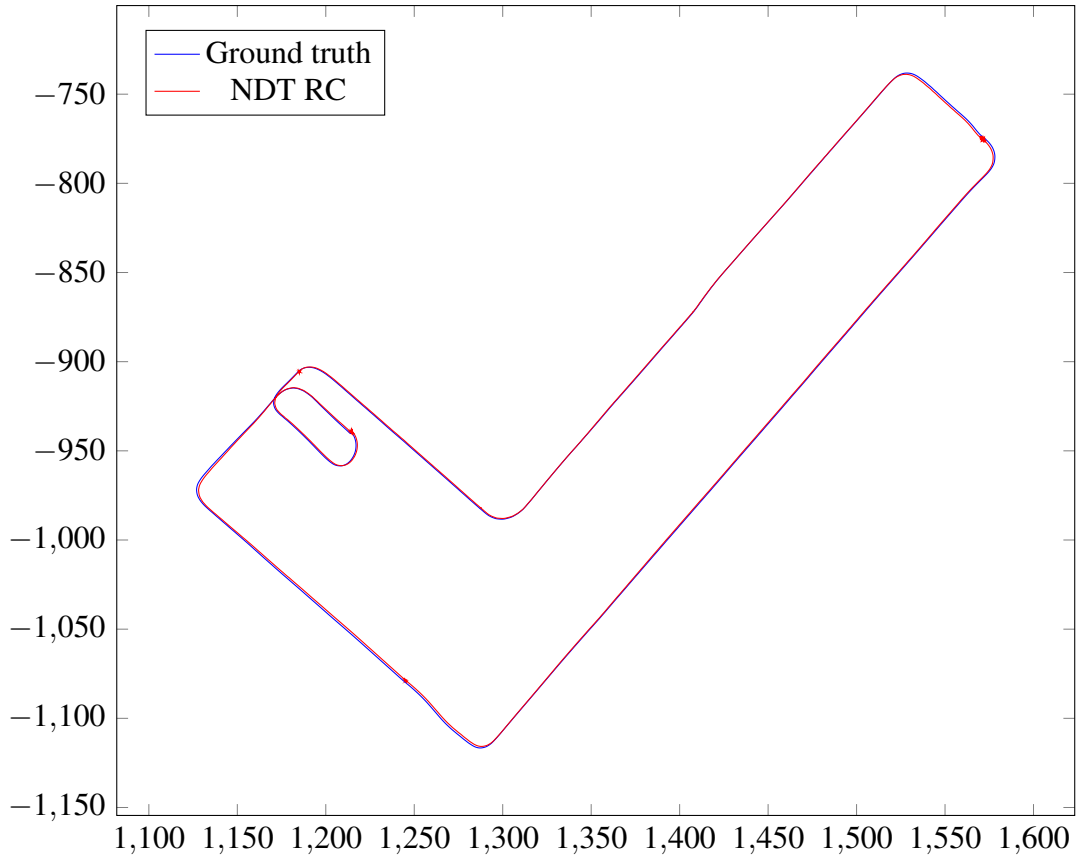


Figure 3.16: Comparison of the trajectory of NDT RC with the ground truth

Table 3.3: Error statistics applied to the ATE for our best result (in meters)

Statistic	Mean	RMS	Median	Std dev	Min	max
Value	1.42	1.49	1.49	0.45	0.55	2.38

Table 3.4: Average runtime detailed for our most accurate result (in milliseconds)

Part	Total	Scan importation	Matching	Map update	Recentring
Duration	229	4	218	5	<1

negligible ($< 0.5\,\%$) compared to the total time.

3.5.3.3 Comparison with other algorithms on the Ford dataset

Finally, a comparison is performed with other algorithms on the Ford dataset. The first one is the NDT OM fusion [5], which is based on NDT OM as well. The algorithm stores the NDT maps on the disk when the robot leaves the zone, and reload them when needed. Note that the Xsens MTi-G IMU mounted on the car is used by the authors to provide an initial guess, while the proposed algorithm does not use any sensor apart from the LIDAR. The authors report a mean ATE of 1.7 m, and a runtime consistently below 2 s, including disk operations. The NDT RC outperforms this algorithm in ATE (1.42 m), and is much faster, with an execution time of around 229 ms (at the same resolution, the runtime would be 360 ms). This difference can be explained by the recentring algorithm, which allows to not perform disk accesses.

Another NDT based algorithm is Segmented Region Growing NDT (SRG-NDT) [86], which adds two main modifications to the traditional NDT distribution to distribution algorithm. First, the ground points are removed from the scans. Second, the remaining points are clustered using a region growing algorithm. Those two operations reduce the runtime and increase the precision compared to D2D NDT. The authors show that NDT D2D can compete on runtime, but loses in precision compared to SRG-NDT. Note that this algorithm performs only registration, no global map is created. The authors report an average error from one scan to the next compared to the ground truth. The results shown in this section are not directly comparable since only the first dataset (around 3800

scans) is used, while they use both datasets (around 7200 scans). The same metric was nonetheless computed with our method. For the SRG-NDT, the authors report a mean translation error of 42 mm and a runtime of 250 ms on both datasets. Computing this metric using the proposed method on only the first dataset gives a mean translation error of 16 mm, and a runtime of 229 ms.

In the work of Tamidji and Ye [87], a method based on the fusion of LIDAR and visual data is proposed. RANSAC is used with an extended Kalman filter. The authors compare their method with generalized ICP [71]. Unfortunately, the only error metric reported is the final position error. This metric reduces the impact of errors at the end of the trajectory and is very trajectory dependent, which is why it was chosen to use the ATE. The final position error reported is 27.93 m for the authors' method, and 16.75 m for generalized ICP. The authors do not report a runtime for their method, but report a runtime of 22 s for the generalized ICP algorithm using a Core i7, with no mention of the specific generation of the CPU. the NDT RC algorithm is more precise: the end of trajectory error is 1.87 m. The proposed algorithm is also faster with a runtime of 229 ms, while using only LIDAR data.

A SLAM method based on the extraction of planar segments from the LIDAR data is proposed by Lenac et al. [88]. The authors report the RMSE on this dataset, however the metric printed in equation (56) is actually the square root of the mean error. It is assumed that this is a printing mistake and that the actual RMSE was computed. The RMSE for their method is 4.48 m. The mean runtime reported are around 250 ms for point cloud segmentation, 1.5 ms for relative pose computation, and less than 50 ms for the global map update. However, because this is a SLAM system, the global map runtime can increase with time. A spike in runtime of more than 600 ms can be seen at the end of the trajectory (figure 33). The proposed algorithm has a lower RMSE of 1.49 m, and a faster runtime of 229 ms. This comparison is interesting because it shows that a lower error is obtained with NDT RC despite the absence of loop closure mechanism.

3.5.4 Evaluation on the KITTI dataset

3.5.4.1 Error metrics

The errors are computed by using the code provided with the dataset [89]. For clarity reasons, the method to compute those errors is described below. The details can be found in the original publication [90].

Let a set $\mathcal{S}^* = \mathbf{T}^*_{i,i\in[\![1,n]\!]}$ be the set of estimated poses, n being the number of poses. Let l_s be a sequence size in meters. Then there is a number k of subset of $\mathcal{S}^*$ whose size is equal to l_s. If there is no such subset, i.e. if the length of the whole trajectory is less than l_s, then the measurement is skipped for this value of l_s. Let those subsets be $\mathcal{L}^{s*}_{j,j\in[\![1,k]\!]}$. Given a subset $\mathcal{L}^{s*}_j$, let $\mathbf{T}^*_f$ be the first pose of this subset and $\mathbf{T}^*_l$ be the last one. For the equivalent subset extracted from the ground truth sequence, those poses are named $\mathbf{T}_f$ and $\mathbf{T}_l$.

The pose error for this subset is computed as:

$$\mathbf{T}^{s,j}_{\text{error}} = \left(\mathbf{T}^{*-1}_f \mathbf{T}^*_l\right)^{-1} \mathbf{T}^{-1}_f \mathbf{T}_l. \tag{3.61}$$

The error is then divided into translational error e_t and rotational error e_r, which are averaged over the length of the subset. Then, this error per meter is averaged over all the subsets $\mathcal{L}^*_j$:

$$\begin{aligned} e_t &= \frac{1}{sk} \sum_s \sum_{j=1}^{k} \frac{\|\text{trans}\left(\mathbf{T}^{s,j}_{\text{error}}\right)\|}{l_s}, \\ e_r &= \frac{1}{sk} \sum_s \sum_{j=1}^{k} \frac{|\angle\text{rot}\left(\mathbf{T}^{s,j}_{\text{error}}\right)|}{l_s}, \end{aligned} \tag{3.62}$$

where $\angle$ is the rotation angle of a rotation matrix. e_t can then be expressed in percentage and e_r in radians per meter.

This is the error for one set of poses. The KITTI dataset includes 11 sets of poses, those errors are then average over those 11 sets to give the final errors that are reported here. As of time of writing, the lengths of the sequences are from 100 m to 800 m by increment of 100 m.

3.5.4.2 Comparison of results on the KITTI dataset

The parameters used with the proposed method on the KITTI dataset are set to the values described in Table 3.2.

The NDT RC is compared to the D2D NDT [75] as implemented on the Github repository of the authors[9]. The parameters for the D2D NDT on the KITTI dataset have been optimised by Zaganidis et al. [91], but those parameters did not result in the best score in our tests. The score obtained with this configuration is an error e_t of 6.0 cm and an error e_r of 1.9×10^{-4} rad. Instead, the following parameters are used: a single 3D matching, a resolution of 0.5 m, a matching neighbourhood size of 2, a map width of 250 m and a map height of 40 m. For both the D2D NDT and the NDT RC, the initial guess for the matching is the result of the previous successful matching.

A recent NDT variation has been proposed with good results on the KITTI dataset: the SE-NDT [91]. This method uses semantic information to improve the matching stage. Two NDTs are created, for edge and planes points respectively. Then the matching is performed between two NDTs of the same type. Since the SE-NDT has been evaluated on the KITTI dataset by using the standard error measures, the results are comparable to ours. Thus, the results of the SE-NDT paper [91] are mentioned here. The runtime is not reported directly since the comparison does not seem valid: in their paper, their algorithm takes 2.82 s while the D2D NDT takes 4.15 s. In our case, the D2D NDT, with the same set of parameters, runs in less than 0.5 s. Such a difference might be explained by hardware or implementation differences, but this is impossible to validate without access to SE-NDT implementation. For this reason, it was chosen not to report this result in

[9] `https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic`, last pull done on Tuesday 27th November, 2018

Table 3.5: Results on the KITTI dataset

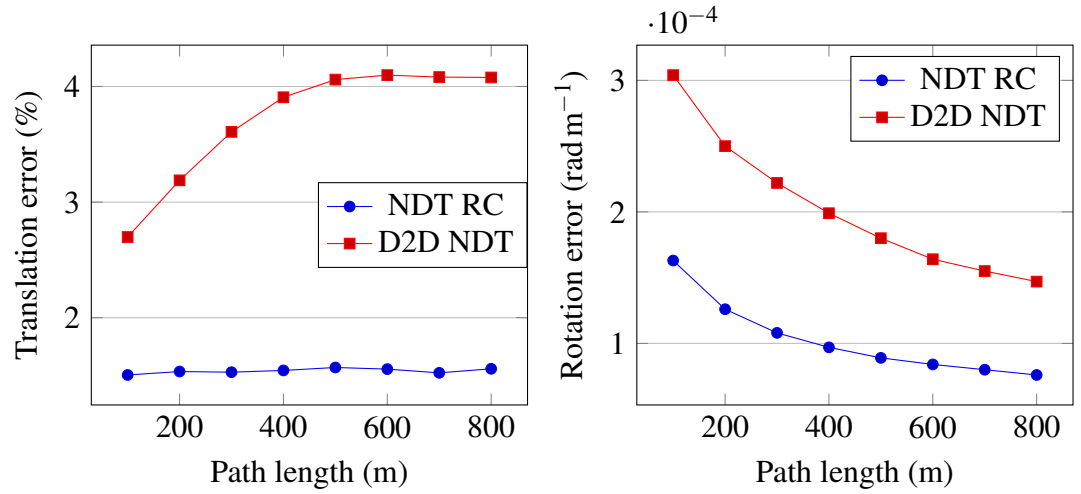
Method	e_t (%)	e_r (mrad m^{-1})	Runtime (ms)
NDT RC	1.54	0.106	187
SE-NDT [91]	2.60	0.2	See comments
D2D NDT [75]	3.65	0.208	499

Table 3.5.

It is important to note that both D2D NDT and SE-NDT are purely odometry methods: they do not maintain a consistent map between each scan, while the NDT RC method does. It is nonetheless fair to compare ourself to those methods since they are applicable to the same scenarios. Indeed, the recentring algorithm makes it possible to move the map with the robot. NDT OM for instance, could not be applied directly to the KITTI dataset because the map would not fit into memory.

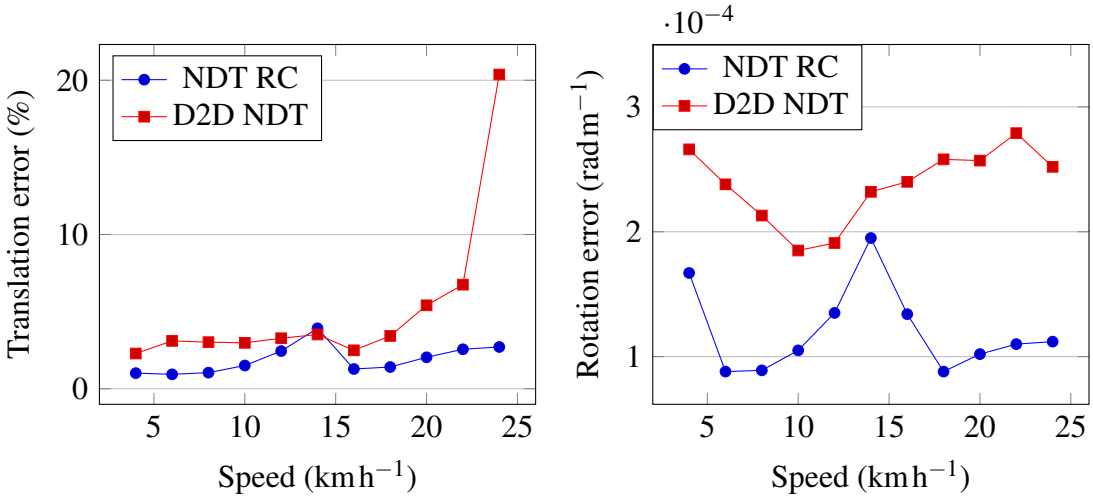
The results are compiled into Table 3.5. The proposed method outperforms both the D2D NDT and the SE-NDT for translational and rotational error while being faster than D2D NDT due to the higher cell size. The average translational and rotational errors are plotted according to the path length and the velocity of the car in Figure 3.17 for our method and D2D NDT. In particular, the proposed method has a lower error when the speed of the vehicle increases. The same graph can be found in the work of Zaganidis et al. [91] for the SE-NDT.

In order to visualise the results, the trajectory of the proposed method and the D2D NDT are plotted against the ground truth in Figure 3.18.



(a) Translation error in function of the path length

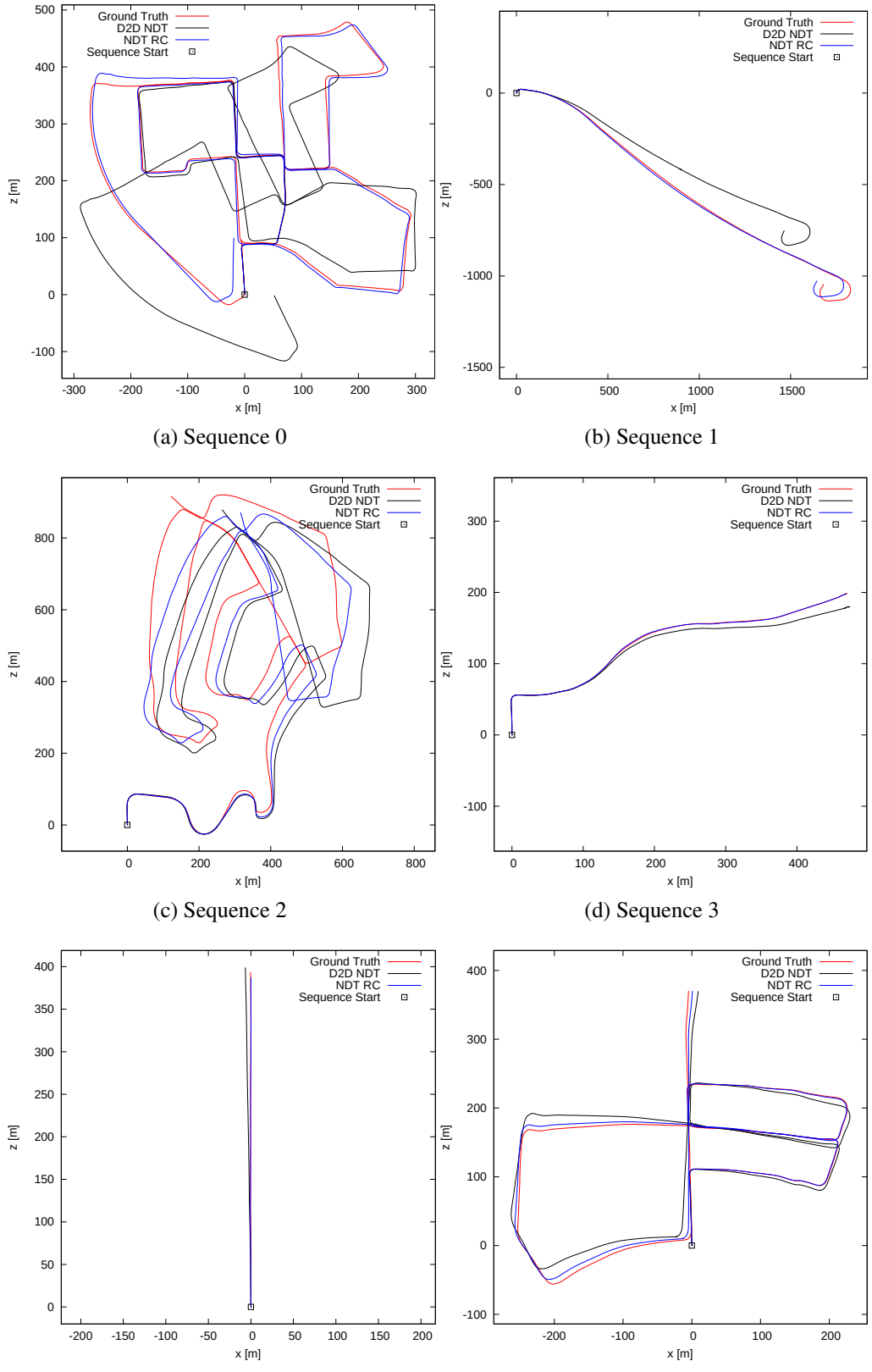
(b) Rotation error in function of the path length



(c) Translation error in function of the vehicle velocity

(d) Rotation error in function of the vehicle velocity

Figure 3.17: Average translation and rotation error for the 11 sequences



(a) Sequence 0

(b) Sequence 1

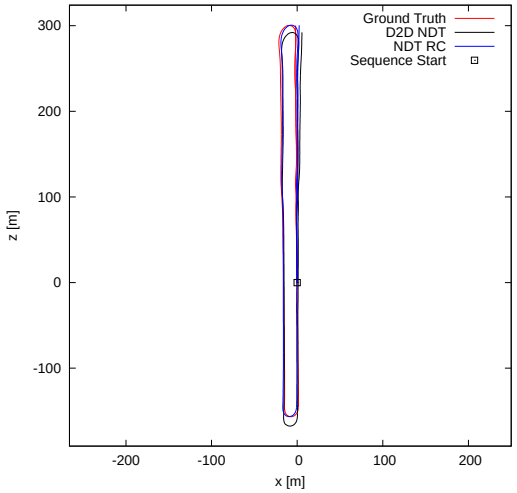
(c) Sequence 2

(d) Sequence 3

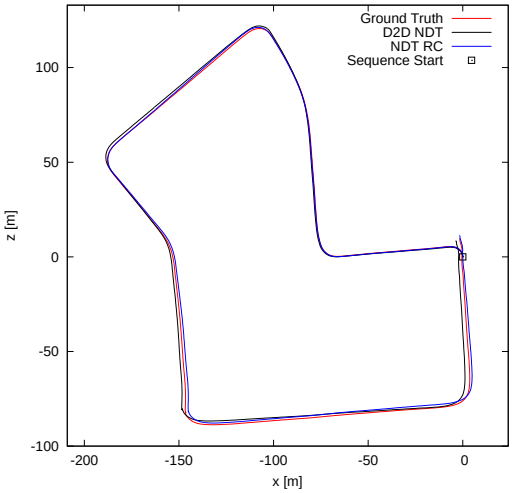
(e) Sequence 4

(f) Sequence 5

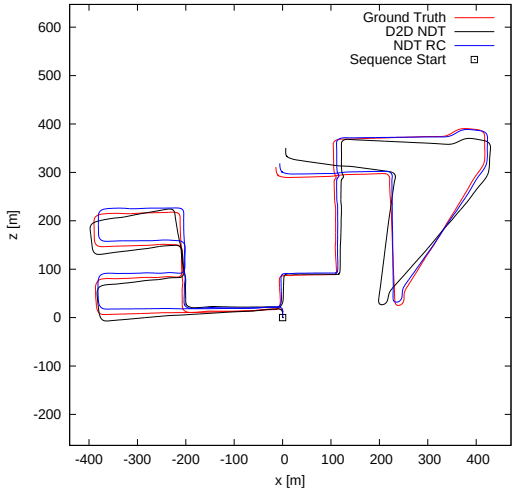
Figure 3.18: Trajectories on the KITTI dataset



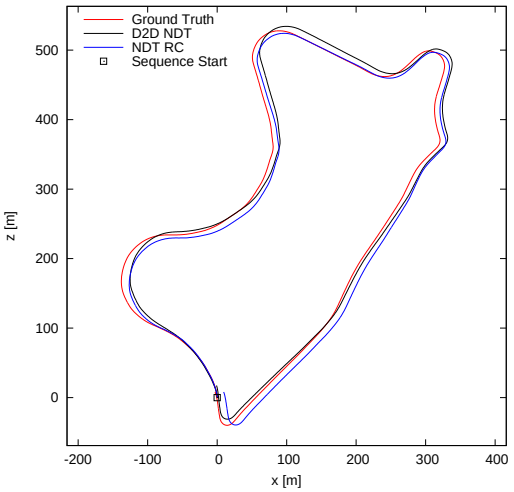
(g) Sequence 6



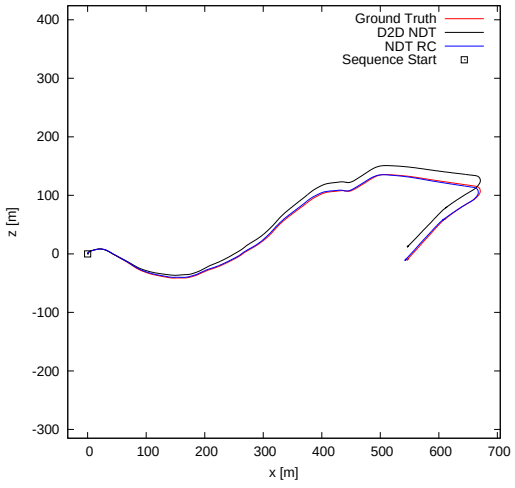
(h) Sequence 7



(i) Sequence 8



(j) Sequence 9



(k) Sequence 10

Figure 3.18: Trajectories on the KITTI dataset

3.6 Conclusion

In this section, NDT RC, a mobile mapping algorithm based on the NDT OM has been presented. This algorithm introduces a novel recentring mechanism that allows unlimited movement of the robot and a new raytracing implementation. The implementation of those two features has been discussed. The performance of the proposed algorithm has been validated using the odometry metric on two public datasets. NDT RC was found to outperform comparable methods using LIDAR data on those datasets.

Since the proposed algorithm implements a recentring of the map, it is compatible with algorithms that improve the NDT matching stage, such as SE-NDT. It is likely that such a combination could lead to even better results in future works.

The mapping algorithm developed in this chapter fulfils the requirements enunciated in Section 3.1.1. The representation of the environment can now be used to implement an obstacle avoidance algorithm that considers the shape of the obstacles. Before investigating this new obstacle avoidance algorithm, the sensor used to gather the data is considered. So far, a LIDAR has been used, which suffers from a density problem. This issue will be investigated in the next chapter.

Chapter 4

Fusion of LIDAR and stereo data for depth map

Contents

In the last chapter, the only sensor used to generate the map is a LIDAR, which provides precise point clouds but lacking of density. It is proposed here to increase the density of the point cloud obtained from the LIDAR using stereo cameras.

In this chapter, an algorithm to compute dense depth maps is proposed. It operates by fusing information from a stereo camera and a LIDAR. The strategy is to estimate reliably the disparities of a sparse set of points, then a bilateral filter is adopted to interpolate the missing disparities. Finally, the interpolation is refined. The proposed method is tested on the KITTI dataset and is compared against several other methods which fuse those modalities, or are extended to perform this fusion. Those tests show that the proposed method is well performing in comparison with other fusion methods. The influence of this fusion scheme on the mapping process is analysed.

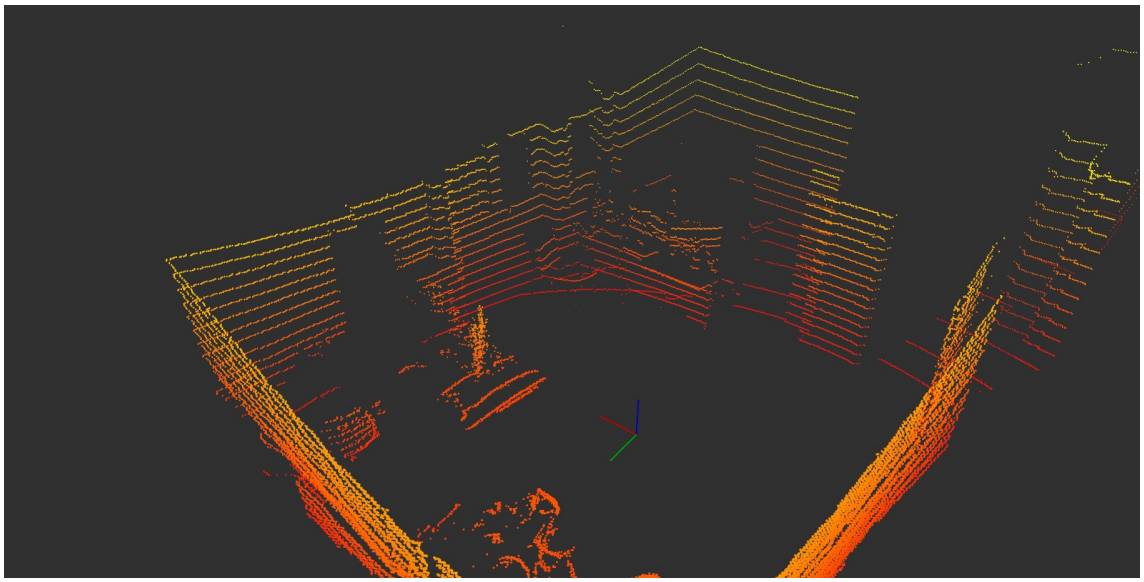
4.1 Introduction

In order to avoid obstacles, it is first necessary to detect them. This chapter investigates a method to increase the number of 3D points available for the obstacle avoidance system. The objective is to obtain a denser point cloud describing the obstacles around the UAV. A denser point cloud means a more robust obstacle avoidance since it increases the chances to detect an object on the way of the robot. This is particularly important in the context of UAVs with a limited payload, which cannot carry the bulkiest LIDARs, meaning that the point cloud available to the small UAVs from a LIDAR sensor are sparse.

In order to illustrate qualitatively the problem, a picture of an office is presented in Figure 4.1 along with the LIDAR scan extracted from the VLP-16, visible in the picture.

This LIDAR is an acceptable sensor to mount on a UAV given its light weight and small

(a) Picture of an office, note the VLP-16



(b) Point cloud from the VLP-16

Figure 4.1: Example of a scene and the associated LIDAR scan from a VLP-16

size. Although the VLP-16 uses 16 beams, its limited vertical range does not allow the

Table 4.1: Comparison of sensors for obstacle detection

Type of sensor	Advantages	Drawbacks
Camera	• Dense output • Lightweight • Cheap	• Need stereo to get depth • Need a matching algorithm to extract obstacles • High sensitivity to illumination changes and images blur
LIDAR	• Precise output • Works at night • Output can be used directly	• Sparse output • Expensive • Can be heavy
Ultrasonic sensor	• Can detect transparent materials • Lightweight • Cheap	• Short range • Low resolution in cluttered environment obstacles
Radar	• Long range • Robustness to weather conditions such as bright sun and fog	• Heavy and power hungry • Low resolution in cluttered environment obstacles

detection of the side of the desk to the left of the picture. Moreover, the limits of the screens on the desk facing the LIDAR are ill defined. The planar surface of the desk facing the LIDAR is not clearly detected as well. Such issues could be sorted by using stereo cameras to obtain a denser point cloud.

In order to justify the choice of sensors in this thesis, the possible sensors usable for detecting obstacles are examined in the next section.

4.1.1 Sensors for obstacle detection

There are several possible sensors to detect obstacles around small UAVs. A comparison between those sensors can be found in Table 4.1.

Since the obstacle detection algorithm should work indoor as well as outdoor, RGBD sensors are excluded from this comparison because the sunlight makes them difficult to use outdoor [92]. The sensors commonly used for 3D perception are cameras and LIDAR scanners, which can be mounted on small UAVs [25]. Cameras are cheap, lightweight and provide a dense output. However, depth retrieval out of a system of stereo cameras

is non trivial [93]. Indeed, matching between two images can be challenging because of uniform areas, illumination changes or blur. On the other hand, LIDARs compute depth directly, but the output is more sparse compared to depth retrieved from cameras. Thus, it makes sense to pair cameras and LIDARs, because they complement each others. Recent progress in LIDAR technology allows the use of light 3D LIDAR scanners (590 g) providing 300000 points per second [94]. It then becomes realistic to include this kind of sensor on small UAVs.

4.1.2 Objective and prior work

The general idea to fuse LIDAR and camera data is often to use the LIDAR scan to reduce the disparity range when searching for stereo camera matches. This way, less false matches occur in the images. However, because the LIDAR scans are sparse, camera images are used to interpolate the missing disparities. In the following, precise 3D points will be referred to as support points. Those can come form LIDAR data or robust stereo matching.

LIDAR scans can be used to improve the Semi Global Matching (SGM) algorithm [95] by narrowing the disparity range [96]. In this method, a penalty is added in the energy function for disparities which differ from the LIDAR measurements. Although the algorithm is shown to be faster than the original SGM, an absolute framerate is not provided. Moreover, while qualitative improvements are shown, there is no quantitative comparison.

In the work of Maddern and Newman [97], the stereo framework of the Efficient Large Scale Stereo Matching (ELAS) algorithm [89] is extended to take into account LIDAR points as new support points. A new post processing step is also introduced to obtain a fully dense result. The algorithm is implemented on a GPU and provides an estimation of the depth uncertainty. In [98], conditional random fields are used in conjunction with a color segmentation of the image to get the depth map, but the runtime is not specified.

In the work of Andreasson, Triebel, and Lilienthal [99], the depth interpolation be-

tween support points is done by weighting each depth depending on its position in the image and pixel intensity. This algorithm is tested on different indoor and outdoor datasets, but those datasets are not public.

Premebida et al. [100] use a bilateral filter to compute a depth map from the LIDAR measurements alone. A clustering algorithm is used on each pixel to better detect discontinuities. Only pixels in the same cluster are fed to the bilateral filter. Out of the 142 scans available in the KITTI dataset [101], only 100 are used for testing.

The first objective of this chapter is to generate a dense depth map in real time using a combination of LIDAR and stereo measurements. The context is obstacle avoidance for UAVs. Thus, it is more important to precisely detect close objects compared to further obstacles. The work from Geiger, Roser, and Urtasun [89] is extended in a similar manner to [97] in the sense that LIDAR points are added to the list of support points, but the interpolation method is modified to a bilateral filter inspired by Premebida et al. [100]. The bilateral filter should handle the discontinuities in the depth profile better than a linear interpolation.

Indeed, the challenge in depth estimation lies in the discontinuities: all the algorithms listed in the review above, except Badino, Huber, and Kanade [96], operate by starting from points where disparity is known, the support points. They then differ by the method used to interpolate the missing disparities. This interpolation is more or less difficult depending on the number of support points, and the presence of discontinuities. In Geiger, Roser, and Urtasun [89] and Maddern and Newman [97], the interpolation is linear, and the discontinuities are handled by keeping track of the disparities around these support points. In the refinement phase, if one of the neighbouring disparity is a better fit, it is chosen. This allows a discontinuity detection. Testing conducted in this chapter indicates that adding this feature improves the depth estimation, especially in the foreground. Another way of handling discontinuities is to modify the filter itself: one of the property of the bilateral filter is to preserve discontinuities, as demonstrated in Badino, Huber, and Kanade [96]. However, the drawback is the need for a fixed window, thus the algorithm

has to be changed for different situations (especially different image sizes). The idea is to modify the bilateral filter to always use 3 points, computed by a Delaunay triangulation. This way, the algorithm naturally adapts to the presence or absence of support points: if there is an area with close support points, then the Delaunay triangles are small, and the depth estimation should be precise. On the other hand, if the support points are far from each other, then the filter is less precise, but an estimation is still computed. This can be seen as an adaptive bilateral filter, to which is also added a color component similar to the work of Andreasson, Triebel, and Lilienthal [99].

The second objective of this chapter is to evaluate how different algorithms can benefit from the fusion of LIDAR and stereo. In order to do a comparison, the KITTI benchmark [101] was used for stereo, LIDAR and fusion of stereo and LIDAR. While this work has already been carried out for the work of Maddern and Newman [97], this comparison is performed on the proposed algorithm and the methods of Premebida et al. [100] and Geiger, Roser, and Urtasun [89] with two metrics (percentage of erroneous pixels and root mean square error (RMSE)). Depth maps are often evaluated on a dataset, but the results are difficult to compare since the datasets might be different, and the metrics used can change. For instance, while Maddern and Newman [97], Miksik et al. [98], and Premebida et al. [100] are all evaluated on the KITTI dataset, the error metrics used are different. Miksik et al. [98] use Root Mean Square Error (RMSE), while Maddern and Newman [97] and Premebida et al. [100] use the percentage of erroneous pixels. Moreover, Premebida et al. [100] uses only a part of the full dataset. Those examples aim to show that even when the same dataset is used, it can be difficult to compare results. This chapter aims to provide a comparison, in similar conditions, between several of those algorithms. In order to judge the benefits from fusion, some algorithms designed for a single modality (such as Geiger, Roser, and Urtasun [89] for stereo and Premebida et al. [100] for LIDAR) are extended to the fusion of dual modalities.

Some background on stereo vision is proposed in Section 4.2 to introduce the basics of the proposed method. The fusion algorithm is detailed in Section 4.3, and testing of

the methods takes place in Section 4.4.

4.2 Background on stereo vision

4.2.1 Stereo vision and depth

It is not possible to recover the depth of an object from a single camera alone: there is a scale ambiguity [102]. This is however possible with two cameras, given that the intrinsic parameters of the cameras are known, and that their relative position is known as well [102]. Those information can be obtained by calibrating the stereo system of cameras.

Former methods to calibrate cameras were difficult to perform [103], however calibration is now possible using only simple patterns [104]. Calibration a stereo system can be performed by taking several pictures of a known planar pattern, without any other information needed [105]. There are available tools in OpenCV [106] and Matlab [107] to perform the calibration.

Once the calibration parameters of the stereo system are known, it is possible to rectify the images [102], which means that:

- The two images are aligned on the same plane.
- The focal lengths of both cameras are the same.
- The two images are vertically aligned. This means that if a particular pixel is on line l of the left image, the same pixel will be observed on the same horizontal line l in the right image. This has important implications for the processing: to perform a matching between both images, there is no need to look in the whole other image, but just on one line, which massively reduce the number of computations. This can be seen on Figure 4.2 which shows two rectified images from the KITTI dataset [101] acquired from the left and right camera. Red horizontal lines are

Figure 4.2: Left and right images from the KITTI dataset after rectification: the horizontal lines are aligned in both images

plotted on those images. It can be verified that if an pixel is on a given line in one image, it will be located on the same line in the other image.

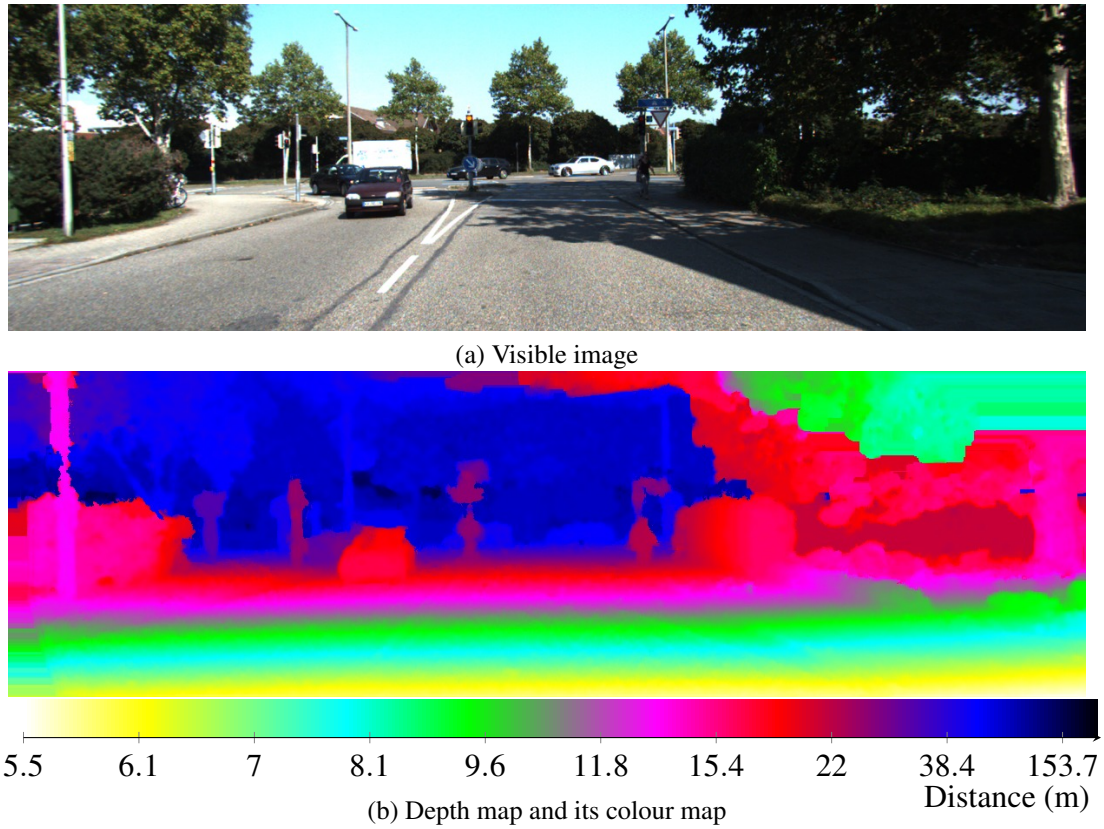
A depth map is an image, where each pixel encodes the depth of the corresponding object, and not the light intensity. An example of depth map and its visible counterpart is shown in Figure 4.3. For instance, if the object is a vertical plane with respect to the camera, it should appear in a uniform color in the depth map. From a depth map, knowing the calibration parameters, it is possible to backproject the pixels in 3D (since the depth is available) to create a point cloud.

The problem is then to compute depth. Let us consider a stereo pair of rectified images $\mathcal{I}_l$ and $\mathcal{I}_r$. Let a feature be visible into both images, at coordinates (x_l, y) in the left image and (x_r, y) in the right image. Then the disparity d for this feature is the horizontal shift of the feature between the left and right image:

$$d = x_l - x_r. \tag{4.1}$$

The disparity is usually expressed in pixel. Let us call f the focal length of the camera, and b the baseline (i.e. the distance between both cameras in a stereo setup). Then the disparity can be linked to the depth Z by the following equation:

$$d = \frac{bf}{Z}. \tag{4.2}$$



(a) Visible image

(b) Depth map and its colour map

Figure 4.3: Example of depth map

This means that knowing disparity means knowing depth. Due to the inverse relationship between both, it also means that when the object comes too far compared to the baseline and focal length, the difference in disparity becomes too small. Thus, a stereo system is only suitable for distances comparable to the baseline, which is not an issue at close range. The depth map computation then becomes a matching problem: to find the depth of a pixel, its position in the other image has to be found.

4.2.2 Methods for depth map computation: local, global, semi global

The methods to solve this correspondence problem can be classified in two broad categories: global methods and local methods. The main difference is that local methods do not directly enforce smoothness on the disparity over the depth map, while global methods do. Local methods usually deal with a window around each pixel, which makes them

fast, but the quality of the result depends on the size of the window. As a result, texture less areas cannot be reliably matched. On the other hand, global methods try to minimise an energy function over the whole image while incorporating smoothness constraints. Let $I(D)$ be a disparity image (i.e. a depth map). A global method would try to minimise the following energy function [95]:

$$E(D) = \sum_{p \in P} \left(C(p, D_p) + \sum_{q \in N_p} \mathrm{PT}\left(D_p - D_q\right) \right), \tag{4.3}$$

where P is the set of all pixels, $C(p, D_p)$ is the matching cost of pixel p assuming a disparity D_p (typically the Euclidean distance between two descriptors), N_p is the neighbourhood of pixel p. PT is a penalty cost depending on the difference in disparity between pixel p with disparity D_p and pixel q with disparity D_q, where q is in the neighbourhood of p. The matching cost and penalty cost depend on the method used. The goal of the penalty cost is to ensure smoothness of the disparity: in order for the disparity to change a lot between two neighbouring pixels (i.e. if one object is a lot behind the other), the cost becomes higher.

The issue is that finding the minimum for this function in two dimensions is a NP complete problem [108], thus approximations have to be used. Global methods are usually computationally heavy, and require a lot of memory (several GB for a 1 MB image).

A hybrid between global and local methods is Semi Global Matching (SGM) [95]. The idea is that, instead of minimising the energy in equation (4.3) over the whole image, it is only done along one dimensional paths, and the result is then summed over all paths. The modified energy equation is:

$$E(D) = \sum_{p \in P} \left(C(p, D_p) + \sum_{\substack{q \in N_p \\ ||D_p - D_q| = 1}} \mathrm{PT}_1 + \sum_{\substack{q \in N_p \\ ||D_p - D_q| > 1}} \mathrm{PT}_2 \right). \tag{4.4}$$

The difference is twofold: there are two penalty functions, one for the small disparity changes (1 pixel, PT_1), and the other for the bigger changes (PT_2). Note that PT_2 can depend on the intensity of p and q. The second difference is that this energy is only evaluated along a 1D paths, i.e. lines instead of the whole 2D images. The number of paths can change, the more there are, the closer it goes to global matching, and the slower it is. SGM is a popular algorithm providing precise results (see the website of the KITTI dataset [101] for benchmarks), it is however slow on the CPU. Some GPU implementations do allow real time performances [109, 110]. For this reason, a different technique is investigated, which does not rely on the minimisation of an energy function, but on interpolation.

4.3 Fusion of LIDAR and stereo

The depth estimation takes place in two phases, as in the work of Geiger, Roser, and Urtasun [89].

Prior estimation: Use the stereo image and the LIDAR scanner to acquire a sparse set of points of known depth. Then, use interpolation to obtain an estimation of the depth in the rest of the image.

Refinement of the estimation: The image is used to refine the depth estimation.

The Bayesian framework used is introduced by Geiger, Roser, and Urtasun [89], it is explained here for clarity purposes and modified to take color into account. A support point $\mathbf{s}_m$ is defined as the concatenation of its image coordinates (u_m, v_m), its disparity d_m, the standard deviation σ_m of this disparity, and the image intensity vector $\mathbf{I}_m$ of this pixel. Let $\mathbf{S}$ be a set of support points. For any pixel p_n, an observation $\mathbf{o}_n^{(l)}$ (for the left image, and $\mathbf{o}_n^{(r)}$ for an observation in the right image) is the concatenation of the coordinates (u_n, v_n) of this pixel, a descriptor $\mathbf{f}_n$ of this pixel, and the image intensity vector $\mathbf{I}_n$ of this pixel. For an observation $\mathbf{o}_n^{(l)}$, the observations on the same horizontal lines in the right

image are $\mathbf{o}_1^{(r)}, \ldots, \mathbf{o}_N^{(r)}$ (abbreviated $\mathbf{O}^{(r)}$). The disparity d_n^* for a given observation $\mathbf{o}_n^{(l)}$ is computed according to a maximum a-posteriori estimation:

$$d_n^* = \text{argmax}\, p(d_n|\mathbf{o}_n^{(l)}, \mathbf{o}_1^{(r)}, \ldots, \mathbf{o}_N^{(r)}, \mathbf{S}). \tag{4.5}$$

In order to compute the prior, the identity proved in appendix B.2 is used:

$$p(d_n|\mathbf{o}_n^{(l)}, \mathbf{o}_1^{(r)}, \ldots, \mathbf{o}_N^{(r)}, \mathbf{S}) = \frac{p(d_n|\mathbf{o}_n^{(l)}, \mathbf{S}) p(\mathbf{O}^{(r)}|\mathbf{o}_n^{(l)}, d_n, \mathbf{S})}{p(\mathbf{O}^{(r)}|\mathbf{o}_n^{(l)}, \mathbf{S})}. \tag{4.6}$$

The observations in the right image are independent from the support points since they are chosen to be on the same epipolar line as $\mathbf{o}_n^{(l)}$. The posterior can then be factorized as:

$$p(d_n|\mathbf{o}_n^{(l)}, \mathbf{O}^{(r)}, \mathbf{S}) \propto p(d_n|\mathbf{o}_n^{(l)}, \mathbf{S}) p(\mathbf{O}^{(r)}|\mathbf{o}_n^{(l)}, d_n). \tag{4.7}$$

The image likelihood can be simplified because knowing $\mathbf{o}_n^{(l)}$ and d_n gives a single $\mathbf{o}_i^{(r)}$ with a non zero probability, thus:

$$p(d_n|\mathbf{o}_n^{(l)}, \mathbf{o}_1^{(r)}, \ldots, \mathbf{o}_N^{(r)}, \mathbf{S}) \propto p(d_n|\mathbf{o}_n^{(l)}, \mathbf{S}) p(\mathbf{o}_i^{(r)}|\mathbf{o}_n^{(l)}, d_n) \tag{4.8}$$

where $p(d_n|\mathbf{o}_n^{(l)}, \mathbf{S})$ is the disparity prior, and $p(\mathbf{o}_i^{(r)}|\mathbf{o}_n^{(l)}, d_n)$ is the image likelihood.

4.3.1 Disparity interpolation and prior computation

To compute the stereo support points, the technique described in the code released with [89] is used: the descriptor $\mathbf{f}$ is a concatenation of Sobel descriptors [111, 112] over a 5×5 window. More precisely, the horizontal and vertical kernels $\mathbf{G_x}$ and $\mathbf{G_y}$ are convoluted

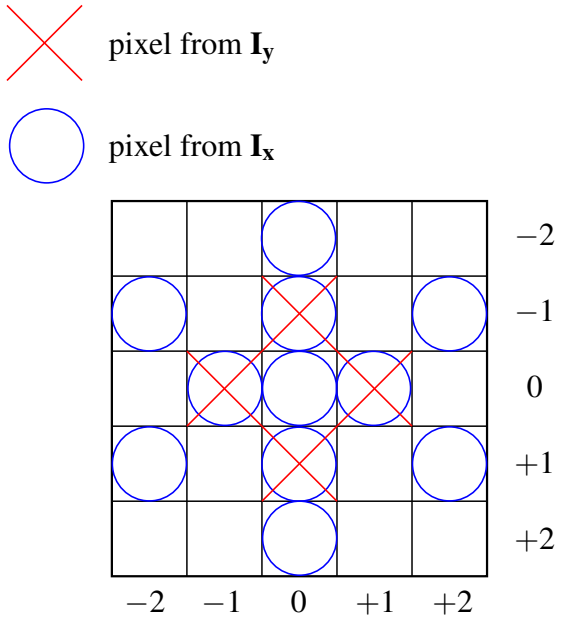


Figure 4.4: Formation of the descriptor used for ELAS matching. The centre of the grid is the considered pixel, the descriptor is formed from adjacent pixels in $\mathbf{I_x}$ (marked with a circle) and $\mathbf{I_y}$ (marked with a cross). Note that the centre is selected twice.

with the original image:

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{4.9}$$

$$\mathbf{G_y} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}. \tag{4.10}$$

This convolution produces a horizontal and vertical response image $\mathbf{I_x}$ and $\mathbf{I_y}$. For each pixel in the original image, a descriptor composed of 16 numbers is computed from a 5×5 window centred on the pixel in $\mathbf{I_x}$ and $\mathbf{I_y}$. The values chosen to populate this descriptor are taken from $\mathbf{I_x}$ and $\mathbf{I_y}$ according to Figure 4.4. The matching takes place on a regular grid using the aforementioned descriptor and the ℓ_1 norm. For each point candidate, a match is searched on the same horizontal line in the right image. Once this

match is found in the right image, a match for this match is sought in the left image. This method is called a left/right check. If the final match is the original point, then this point becomes a support point. The disparity standard deviation is a fixed constant σ_s. In order to generate the LIDAR support points, the LIDAR points are projected into the picture: let $\mathbf{K}$ be the camera projection matrix, and $\mathbf{T}_{lc}$ the transformation matrix from the basis of the LIDAR to the basis of the camera. Let $\mathbf{p_m}^r$ be a 3D position vector in the LIDAR reference frame, then its position $\mathbf{p_m}(u_m, v_m)$ in the image is:

$$\mathbf{p_m} = \mathbf{K}\mathbf{T}_{lc}\mathbf{p_m}^r. \tag{4.11}$$

Then, the corresponding disparity $\hat{d_m}$ is computed as fb/z_m where f is the focal length of the camera, b is the baseline of the stereo system, and z_m is the depth of the LIDAR point in the camera reference frame. The standard deviation σ_m of the support point in the disparity domain is then:

$$\sigma_m = \frac{fb}{z_m^2}\sigma_l \tag{4.12}$$

where σ_l is the standard deviation of the LIDAR range. Both type of support points are accumulated in the same image, if there is an overlap (i.e. there is a LIDAR support point at the same location as a stereo support point), the support point with the lowest disparity standard deviation is chosen. This is a difference with the work of Maddern and Newman [97], where stereo and LIDAR support points are treated separately: by doing this, a more precise interpolation is obtained at the cost of increased computational requirements.

Once the stereo and LIDAR support points are merged, a Delaunay triangulation is performed, separating the image in triangles with support points as vertices. Inside each triangle, the disparity is interpolated. Let $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3$ be the support points at each vertex. In the computation, the range r is used instead of disparity $\hat{d}$, where $r = fb/\hat{d}$. For each support point $\mathbf{s}_{i\in[\![1,3]\!]}$, its position (u_i, v_i), its range r_i, and the image intensity vector $\mathbf{I}_i$

are used during the interpolation. r^*, which is used in the depth kernel of the bilateral filter, is defined as $\min(r_1, r_2, r_3)$, because the targeted application is obstacle avoidance, thus it makes sense to interpolate conservatively (i.e. underestimation of the distance is preferred over overestimation). The same notations, indexed with n instead of i are used to designate the attributes of the inner points of the triangle.

In order to compute the interpolation, a modified bilateral filter inspired from the work of Premebida et al. [100] is used. A traditional bilateral filter replaces an intensity by a weighted average of the intensity of surrounding pixels using different kernels. This principle is applied to the depth interpolation: the bilateral filter estimates the depth of a pixel from the weighted average of the 3 vertices of the triangle that contains the pixel. Three kernel are used: a position kernel (the closest vertex weighs more), a depth kernel (the smallest depth weighs more) and a colour kernel (a similar colour weighs more). Formally, the different kernels for the bilateral filter are defined as follows:

$$G_i^{pos}(u_n, v_n) = \frac{1}{1 + \alpha_{pos}\sqrt{(u_n - u_i)^2 + (v_n - v_i)^2}}, \tag{4.13}$$

$$G_i^{depth}(r^*) = \frac{1}{1 + \alpha_{depth}|r^* - r_i|}, \tag{4.14}$$

$$G_i^{col}(\mathbf{I}_n) = \frac{1}{1 + \alpha_{col}\,\|\mathbf{I}_n - \mathbf{I}_i\|}, \tag{4.15}$$

where G^{pos} is the position kernel, G^{depth} is the depth kernel, and G^{col} is the colour kernel. $\alpha_{pos}, \alpha_{depth}, \alpha_{col}$ are real positive tuning constants. The weights $w_{i\in[\![1,3]\!]}$ are defined, for an inner point p_n:

$$w_i(u_n, v_n, r^*, \mathbf{I}_n) = G_i^{pos}(u_n, v_n) G_i^{depth}(r^*) G_i^{col}(\mathbf{I}_n). \tag{4.16}$$

Using the previous notations, for any point $\mathbf{p}_n(u_n, v_n, \mathbf{I}_n)$ inside the triangle, its range

r_n is computed as:

$$r_n = \frac{1}{\sum_{i=1}^{3} w_i} \sum_{i=1}^{3} w_i(u_n, v_n, r^*, \mathbf{I}_n) r_i. \tag{4.17}$$

This range r_n is then converted back to the disparity estimate $\hat{d}_n(\mathbf{o}_n^{(l)}, \mathbf{S})$. The estimated standard deviation $\hat{\sigma}_n$ is then chosen as the highest among the standard deviations of the 3 support points.

Once this estimate is available, the prior itself is computed as by Geiger, Roser, and Urtasun [89]: the prior is evaluated over the disparity interval $[\![\hat{d}_n - 3\sigma_n, \hat{d}_n + 3\sigma_n]\!]$ as:

$$p(d_n|\mathbf{o}_n^{(l)}, \mathbf{S}) = \gamma + \exp\left(-\frac{(d_n - \hat{d}_n(\mathbf{o}_n^{(l)}, \mathbf{S}))^2}{2\hat{\sigma}_n^2}\right), \tag{4.18}$$

where γ is a real constant. Moreover, the image is divided in 20 pixels wide squares, and the disparities inside each squares are recorded. For an estimated disparity $\hat{d}_n(\mathbf{o}_n^{(l)}, \mathbf{S})$, the union of the interval $[\![\hat{d}_n - 3\sigma_n, \hat{d}_n + 3\sigma_n]\!]$ and the disparities inside the 20 pixels wide square containing the point is called $\mathcal{D}_e$. For any pixel, the prior in equation (4.18) is evaluated over all the disparities in $\mathcal{D}_e$.

4.3.2 Disparity refinement

The image likelihood is computed as in [89]: given an observation in the left image $\mathbf{o}_n^{(l)}$ and a disparity d_n, the image likelihood is for the corresponding right observation (i.e. such that $(u^{(r)}, v^{(r)}) = (u^{(l)} + d_n, v^{(l)})$)

$$p(\mathbf{o}_i^{(r)}|\mathbf{o}_n^{(l)}, d_n) = exp(-\beta [\![\mathbf{f}_n^{(l)} - \mathbf{f}_n^{(r)}]\!]), \tag{4.19}$$

where $\mathbf{f}_n^{(l)}$ is the descriptor in the left image, and $\mathbf{f}_n^{(r)}$ the corresponding descriptor in the right image. β is a real tuning constant.

By inserting equation (4.18) and (4.19) into equation (4.8), and applying a winner takes all strategy to solve equation (4.5), the final disparity d_n^* is obtained for the considered pixel.

The steps to obtain the depth map from a pair of stereo images are summarised in Algorithm 4.1.

4.3.3 Post processing

Several post processing steps are applied on the generated depth map:

1. The depth map is computed for the left and right image, and a left/right check is applied. If the disparities are not consistent between both depth map (with a tolerance), the disparity is considered invalid.

2. Homogeneous disparity zones smaller than a custom threshold are considered invalid since they are unlikely to be correct.

3. A standard bilateral filter is applied with a sliding window of 13 pixels to fill disparity gaps. Estimated disparities and projected LIDAR point cloud are used to interpolate the missing data.

4.4 Evaluation of the quality of the depth maps

4.4.1 Dataset

In order to evaluate the proposed algorithm, the KITTI dataset 2015 [101] is used. This dataset was acquired by a moving car equipped with several sensors. To the best of our knowledge, this is the only dataset which proposes stereo images and LIDAR scans, as well as a disparity ground truth which is the result of a concatenation of several LIDAR scans. The ground truth is formed by taking 5 LIDAR scans before and after the current scene, and projecting them in the current scene. Moreover, for some of the cars which are

Algorithm 4.1: High level proposed algorithm to compute a depth map from a pair of stereo images and a LIDAR scan

Input : left colour image $\mathbf{I}^{(l)}$, right colour image $\mathbf{I}^{(r)}$, LIDAR scan $\mathbf{L}$.
Output: a disparity map of the left image $\mathbf{D}^{(l)}$.

$\mathbf{So}_x^{(l)}, \mathbf{So}_y^{(l)}, \mathbf{So}_x^{(r)}, \mathbf{So}_y^{(r)} \leftarrow$ convolution of $\mathbf{G_x}$ and $\mathbf{I}^{(l)}$, $\mathbf{G_y}$ and $\mathbf{I}^{(l)}$, $\mathbf{G_x}$ and $\mathbf{I}^{(r)}$, $\mathbf{G_y}$ and $\mathbf{I}^{(r)}$; `// Compute descriptors`

Create a left image of descriptors $\mathbf{F}_d^{(l)}$ from $\mathbf{So}_x^{(l)}$ and $\mathbf{So}_y^{(l)}$ using Figure 4.4;

Create a right image of descriptors $\mathbf{F}_d^{(r)}$ from $\mathbf{So}_x^{(l)}$ and $\mathbf{So}_y^{(l)}$ using Figure 4.4;

$\mathbf{S}_l = \varnothing$; `// Get support points from stereo images`

for *all points* $\mathbf{p}$ *on a fixed grid of size 5 in* $\mathbf{F}_d^{(l)}$ **do**

  if *Left/right check of p in* $\mathbf{F}_d^{(r)}$ *is successful with disparity d* **then** $\mathbf{S}_l \leftarrow (\mathbf{p}, d)$;

end

for *all points* $\mathbf{p_m}^r$ *in* $\mathbf{L}$ **do**

  $d = \mathbf{T}_{lc}\mathbf{p_m}^r(2)$; `// Extraction of depth`

  $\mathbf{S}_l \leftarrow (\mathbf{K}\mathbf{T}_{lc}\mathbf{p_m}^r, d)$; `// Add support points from LIDAR`

end

$\mathcal{T} \leftarrow$ Delaunay triangulation of $\mathbf{S}_l$;

for *all* $T \in \mathcal{T}$ **do**

  Get $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$ vertices of T;

  Get r_1, r_2 and r_3 depth of $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$;

  $r^* = \min(r_1, r_2, r_3)$; `// Estimate the prior`

  for *all* $\mathbf{p}_n(u_n, v_n, \mathbf{I}_n) \in T$ **do**

    $r_n = \frac{1}{\sum_{i=1}^{3} w_i} \sum_{i=1}^{3} w_i(u_n, v_n, r^*, \mathbf{I}_n) r_i$;

    $\hat{d}_n = fb/r_n$; `//` f `is the focal length,` b `the stereo baseline`

    Get $\hat{\sigma}_n$ from $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$;

  end

end

for *all points* $\mathbf{p}_n(u_n, v_n, \mathbf{I}_n) \in \mathbf{I}^{(l)}$ **do**

  $\text{Score}_b = +\infty$; `// Initialise the best score`

  for *all* $d_n \in \mathcal{D}_e$ **do**

    Get $\mathbf{f}_n^{(l)}$ from $\mathbf{F}_d^{(l)}$ and $\mathbf{f}_n^{(r)}$ from $\mathbf{F}_d^{(r)}$ using d_n; `// Refine the disparity`

    $\text{Score} = \left(\gamma + \exp\left(-\frac{(d_n - \hat{d}_n(\mathbf{o}_n^{(l)}, \mathbf{S}))^2}{2\hat{\sigma}_n^2}\right)\right) exp(-\beta[\![\mathbf{f}_n^{(l)} - \mathbf{f}_n^{(r)}]\!])$;

    if $\text{Score} < \text{Score}_b$ **then**

      $\text{Score}_b = \text{Score}$;

      $d_n^* = d_n$;

    end

  end

  $\mathbf{D}^{(l)}(n) = d_n^*$;

end

return $\mathbf{D}^{(l)}$;

visible, CAD models are used to increase density. The complete dataset has 200 stereo pairs, although only 142 pairs have an associated LIDAR scan. In the following, the results are evaluated on those 142 pairs of images. Moreover, the *occ* (occluded) ground truth of the dataset is used, which is the ground truth of the online ranking. Finally, a remark on the KITTI dataset: while several LIDAR scans were fused to obtain the ground truth, the average density of the ground truth over the whole dataset was found to be 19.73%. It would be interesting, especially when evaluating algorithms based on LIDAR, to have a denser ground truth.

4.4.2 Metrics

Two metrics are used to evaluate the algorithms:

- The percentage of error (Error_{tot}), i.e. the percentage of pixels which are more than 5% or 3 pixels away from the ground truth. This error is separated between the background (Error_{bg}) and the foreground (Error_{fg}).

- The root mean square error (RMSE), a standard measure of error computed as: $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} \|d_k - d_k^{gt}\|^2}$ where d_k is the predicted disparity and d_k^{gt} is the ground truth.

The density related to the ground truth is also provided (i.e. only the pixels which are both predicted by the proposed algorithm and the ground truth are counted). If some pixels remain undefined (i.e. the density is less than 100 %), the depth map is interpolated by using a method described in the devkit of the dataset [101] (a missing disparity is set to the minimum disparity of its two valid neighbours). The results before this interpolation (i.e. for the original density) are qualified as original, while the results after are qualified as interpolated. In order to ensure a fair comparison, only interpolated results are considered, since the corresponding depth map are fully dense. However, original results are still provided for completeness.

4.4.3 Algorithms and methodology

The aim of this section is to compare several algorithms which deal with a combination of LIDAR and stereo to examine their performance. The following algorithms are compared:

- The proposed algorithm, described in Section 4.3, is referred to as Tri_BF. In order to illustrate the role of the refinement stage, the method with only the prior estimation and the post processing is also included under the name Tri_BF_PO.

- The ELAS algorithm described in [89] by using the provided code, slightly modified to take the LIDAR points into account: the LIDAR points are added as additional support points. Since the proposed method is derived from this framework, it makes sense to compare with it.

- The BF* algorithm from [100], since the proposed algorithm uses a bilateral filter as well. As the results in the paper are only specified for 100 stereo pairs instead of the 142 available, and on the non-occluded set instead of the occluded one (which is the set used by the KITTI website to rank the methods), the described algorithm was replicated with the same parameter's values. The value of the parameter Thr was not provided. It was empirically set to 0.5. The different modalities change the type of support points.

- The fusion scheme described in [97] called "Combined+Pyr" ("Stereo+Pyr" refers to the method with only the stereo points, while "LIDAR+Pyr" refers to the method with only LIDAR points). Given that the experiment was performed in the same conditions (the article mentions 141 out of the 142 image pairs, but one image pair should not change the result in a significant way), the numbers provided in the paper are used. Please note that the interpolated results are not provided. For the fusion case, given the high density of the result (99.62%), the interpolated result is likely to be similar to the original one.

For each pair of pictures, the corresponding LIDAR scan is located, as well as the

appropriate projection matrices, then the LIDAR scan is projected into the current pictures using equation (4.12).

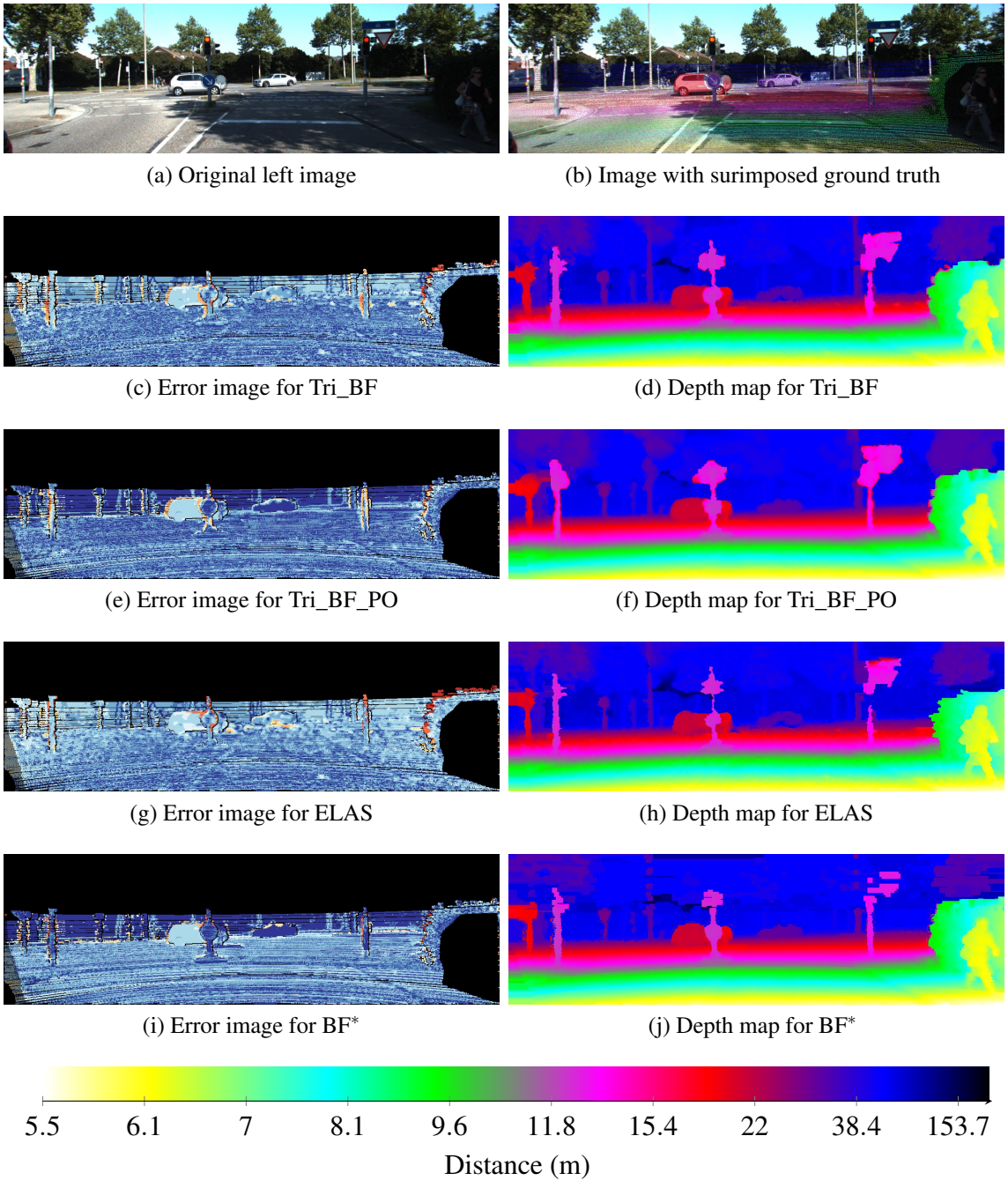
4.4.4 Results and discussion

The quantitative results for the methods mentioned in the precedent section are summarised in Table 4.4.4, for each modality. For the fusion modality, which is the main approach under consideration in this chapter, the proposed method has the lowest RMSE (3.59), and the second lowest error percentage (4.17%) behind BF* (3.51%), which means that it is competitive with other fusion algorithms. Looking at the separation between the background and the foreground shows that the proposed method outperforms the other methods in the foreground, with worse results in the background. This is coherent with the context, which is obstacle avoidance. It can seem surprising that the refinement step actually decreases the percentage of erroneous pixels, but increases the RMSE. A possible cause is that the refinement step more aggressively operate on the boundaries: the depth maps for Tri_BF and Tri_BF_PO in Figure 4.5 shows that the boundaries of foreground objects are slightly better defined for Tri_BF (e.g., for the traffic signs and traffic lights). Even if more pixels are correct, the ones that are incorrect might suffer from a higher error since they are located on a boundary.

As the methods mentioned there are based on interpolation, it makes sense to look at the influence of the density of the support points. Three ways to do that are provided. The first one is a graph showing the influence of the LIDAR point cloud density: Figure 4.6 shows Error_{tot} (for an interpolated depth map, thus 100% density) and Figure 4.7 shows the RMSE for each method, for the fusion of both modalities, when one LIDAR point out of n_{retain} is kept (thus $n_{\text{retain}} = 1$ means all the LIDAR points and $n_{\text{retain}} = 5$ means 20% of them). A second way to look at the influence of the density is to compare, in Table 4.4.4, the fusion with the stereo, or the LIDAR with the stereo, since the LIDAR (and thus the fusion) results in more support points than only stereo. A third way to

Table 4.2: Comparison of different techniques using stereo and LIDAR modalities

Method	Modality	$Error_{bg}$ (in %)		$Error_{fg}$ (in %)		$Error_{tot}$ (in %)		RMSE		Density (in %)
		Interpolated	Original	**Interpolated**	Original	**Interpolated**	Original	**Interpolated**	Original	
Tri_BF	Fusion	3.01	2.62	10.5	9.01	4.17	3.60	**3.59**	3.37	98.59
Tri_BF_PO	Fusion	1.92	1.82	13.2	12.3	3.67	3.42	4.30	4.04	98.55
ELAS	Fusion	3.93	1.28	13.5	5.97	5.42	1.95	3.79	1.98	88.18
BF*	Fusion	1.63	1.53	13.7	11.4	**3.51**	3.00	4.59	4.20	98.07
"Combined+Pyr"	Fusion	NA	4.55	NA	13.18	NA	5.91	NA	NA	99.62
Tri_BF	Stereo	11.2	4.82	14.6	7.25	11.7	5.24	**4.33**	2.53	76.19
Tri_BF_PO	Stereo	17.1	10.9	13.5	6.18	16.6	10.1	4.81	2.80	75.62
ELAS	Stereo	7.83	3.54	16.3	8.43	**9.14**	4.29	4.38	2.43	83.69
BF*	Stereo	14.6	4.99	23.8	6.78	16.0	5.33	5.84	3.08	41.65
"Stereo+Pyr"	Stereo	NA	17.78	NA	16.12	NA	17.51	NA	NA	94.31
Tri_BF	LIDAR	4.27	2.64	9.50	6.69	5.08	3.25	**3.22**	2.49	96.04
Tri_BF_PO	LIDAR	2.28	1.63	14.3	11.3	4.15	3.08	5.46	4.29	97.36
ELAS	LIDAR	13.72	0.311	31.1	2.73	16.4	0.634	7.61	1.02	59.51
BF*	LIDAR	1.59	1.48	15.1	11.9	**3.68**	3.01	4.75	4.24	97.80
"LIDAR+Pyr"	LIDAR	NA	7.17	NA	15.65	NA	8.51	NA	NA	99.24



(a) Original left image

(b) Image with surimposed ground truth

(c) Error image for Tri_BF

(d) Depth map for Tri_BF

(e) Error image for Tri_BF_PO

(f) Depth map for Tri_BF_PO

(g) Error image for ELAS

(h) Depth map for ELAS

(i) Error image for BF*

(j) Depth map for BF*

(k) Colour map for the depth maps

Figure 4.5: Pictures of the depth maps obtained by different methods. Left: the error image indicates how far (from blue, close, to far, red) each pixel is from the ground truth. Right: the depth map shows the estimated depth of each pixel.

estimate visually this influence is to look at the difference between the zone covered by the LIDAR (dense area of support points) and the rest of the picture (sparser area of support points) in the depth maps from Figure 4.5. The BF* method, which uses a

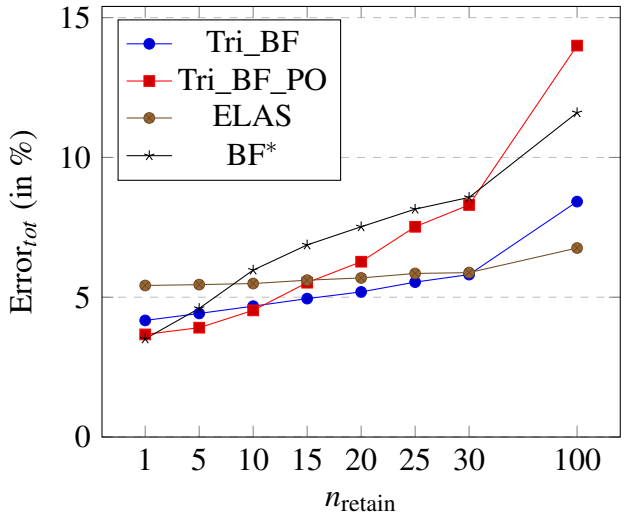


Figure 4.6: Evolution of the error percentage depending on the density of the LIDAR data

fixed window size for the bilateral filter, works well in the fusion and LIDAR modalities (Table 4.4.4), but suffers when the density gets lower (from 3.51% of erroneous pixels to 11.6%, the rate of change can be seen in Figure 4.6). This is apparent in Figure 4.5j, where the results outside the zone covered by the LIDAR (shown in Figure 4.5b) are less precise. This emphasises the need to adapt the size of the window to the density of the input data, such an analysis is provided in the paper of this method [100]. The proposed method uses a similar bilateral filter, but with a dynamic window size (see Section 4.3). Figure 4.6 shows that the refinement step decreases the rate at which the percentage of erroneous pixels increases when the density decreases, while the RMSE remains lower with the refinement (see Figure 4.7). The ELAS method is the less affected by a decrease in density of support points as shown in Figure 4.6, the percentage of erroneous pixels becomes lower than the proposed method when less than around 3.3% of the LIDAR points are retained. Qualitative results are provided in Figure 4.5.

The average computation time, on a computer with an Intel Core i7 6700 processor and 16 GB of RAM is detailed in Table 4.3. Due to the independent computation for each pixel, the proposed algorithm could be parallelised (the framework used, based on the work of Geiger, Roser, and Urtasun [89], has been implemented on a GPU [97]). This same analysis is not carried out on other method since the runtime is heavily influenced

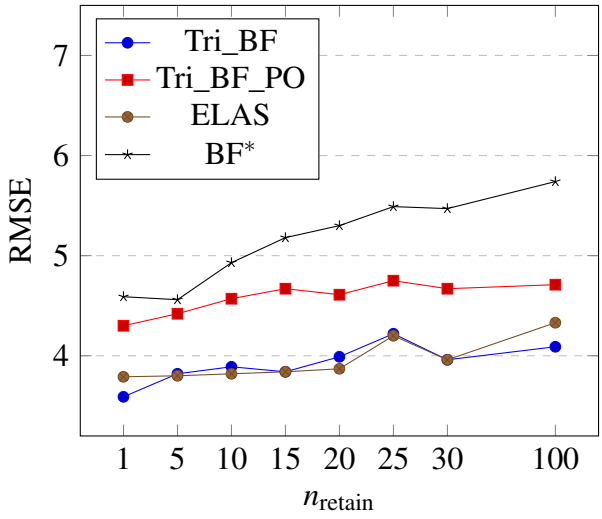


Figure 4.7: Evolution of the RMSE depending on the density of the LIDAR data

Table 4.3: Processing time of the main operations

Operation	Processing time (in ms)
Computation of support points	21
Delaunay triangulation	27
Priors computation	39
Matching	248
Post processing	347
Total	682

by the time spent optimising the program.

As a summary of those results, it has been shown that a high density of support points makes an approach based on bilateral filtering (BF*, Tri_BF_PO) outperform a linear estimation (ELAS). However, a linear estimation appears to be less affected by a reduction in the number of points provided by the LIDAR. A hypothesis to explain this result is that a linear interpolation does not depend on the distance between the points that are interpolated. A bilateral filter however loses in precision when the triangle grows: the strength of the bilateral filter, which is a sensitivity to discontinuity, means that the value of the weights has to decrease quickly between the support points. This suggests lesser performances when the support points are spread too far. Tri_BF, based on a bilateral filter with refinement, is less sensitive to the decrease in density than a standard bilateral filter but more than a linear interpolation. At maximal density, the proposed method is

competitive with the other methods presented (lowest RMSE and 0.66% of erroneous pixels more than the best performing method on this metric, BF*).

Those results show the influence of the fusion scheme on the depth map (or equivalently, point cloud) generated from the sensors. In the next section, the influence of this new point cloud on the mapping process is examined.

4.5 Influence of the sensor fusion on the mapping process

The influence of the fusion scheme on the mapping process is now examined according to two criteria: density of the point cloud and precision of the points.

Three ways of adjusting the density of the generated point cloud are considered. The first one, subject of this chapter, is to increase the density with interpolation methods, at a potential cost of precision. The second one is to set an upper depth limit on the depth map extracted from the stereo cameras: the precision of the depth estimated from a stereo pair of cameras degrades when the depth increases. Thus, reducing the upper limit on the depth of the points generated from the stereo cameras is likely to yield a more precise point cloud, at the cost of density, since less points are obtained. In both cases, there can be a trade-off between density and precision of the point cloud: increase in density may come with a cost of precision. The third option is to use a denser LIDAR which often increases costs, weight and power consumption.

The reason to increase density of the point cloud is to detect more obstacles and to increase the level of details of detected obstacles. Yet, considering the balance between density and precision: the input point cloud has to be dense enough for the obstacles to be detected while keeping a reasonably accurate map. The objective of this section is to provide some qualitative and quantitative information on this trade-off.

It is important to note that the Velodyne HDL-64E LIDAR[1] used in the KITTI [84] datasets is adapted to automotive applications, but not to small UAVs: its weight is around

[1] `https://velodynelidar.com/hdl-64e.html`

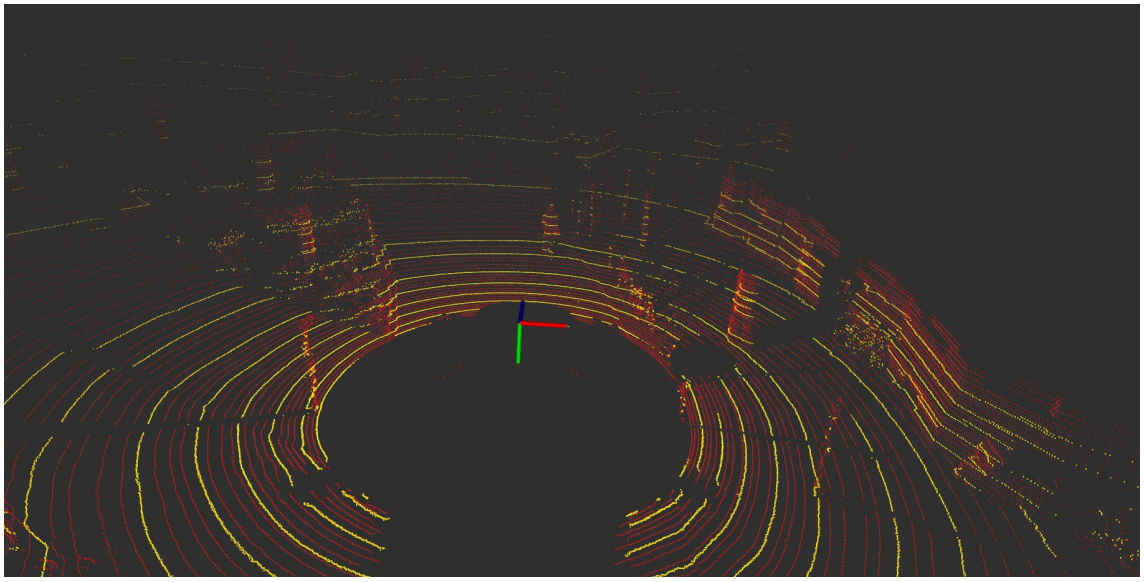


Figure 4.8: LIDAR scan acquired from a Velodyne HDL-64E (in red) and a downsampled version matching the characteristics of the Velodyne VLP-16 (in yellow)

13 kg and it consumes approximately 50 W of power. A more realistic sensor for small UAVs would be the Velodyne VLP-16 lite [94] which weighs 590 g and draws around 8 W of power. The drawback is that the VLP-16 has only 16 vertical beams, compared to the 64 of the HDL-64E, with a vertical resolution going from 0.4° down to 2°. In order to simulate a VLP-16 output from the HDL-64E, a downsampling method is implemented. Since the VLP-16 has a vertical resolution of 2° for a total vertical field of view of 30°, it is not possible to match exactly the vertical field of view of the HDL-64E, which is 25.2°. It was chosen to respect the vertical resolution of the VLP-16, thus the downsampled cloud has only 13 beams, meaning that the point cloud obtained from an actual VLP-16 would be denser compared to the results presented in this section. The downsampling result is illustrated in Figure 4.8.

The fusion scheme for stereo and LIDAR data is the algorithm labelled Tri_BF in this section. The post processing steps are limited to the left/right check.

4.5.1 Density: qualitative assessment

In this section, the advantage of improving the density of the input point cloud is investigated. Such an advantage is difficult to quantify without a ground truth for the map. For this reason, it is proposed to provide a qualitative assessment by examining the point cloud and NDT maps provided by different combination of sensors.

Two possible methods to create a point cloud are considered: the LIDAR can be used alone or associated with the stereo data. Moreover, the LIDAR scan can be the full scan (64 beams) or the downsampled scan (13 beams). Combining those two means that there are four possibilities for creating a point cloud: a full LIDAR scan using LIDAR only (called FullLi in this section), a full LIDAR scan using LIDAR and stereo data (called FullLS in this section), a downsampled LIDAR scan using LIDAR only (called DownLi in this section) and a downsampled LIDAR scan using LIDAR and stereo data (called DownLS) in this section.

The point clouds and NDT maps created from those four combinations are visually examined in this section, using the KITTI dataset. The resolution of the NDT map is set to 0.5 m in order to capture the details of the environment accurately. The objects examined are located between 5 m and 10 m from the sensors, in order to reflect what a real UAV would need to capture in the context of small spaces. As a result, the range of the depth data is limited to 15 m to improve the clarity of the presented pictures. The focus is on small and medium obstacles since they are the most difficult to detect.

The first scene is shown in Figure 4.9, it contains several small poles and a taller pole holding a sign.

Figure 4.10a and 4.11a show the LIDAR scan and the projected depth map from the full and downsampled LIDAR respectively. It can be seen that the downsampled LIDAR point cloud does not cover the small poles. Due to a limit of vertical angles, both LIDARs do not cover the road sign. The depth maps however bring points onto the poles and the sign, both with the full and downsampled version of the LIDAR. Considering the full LIDAR scan, the addition of the stereo data (i.e. the passage from Figure 4.10b to

Figure 4.9: Scene 1

Figure 4.10c) extends the Gaussians on the floor, which allow them to better reflect the shape of the scene. Moreover, the addition of the depth map allows the road sign, non covered by the LIDAR scan, to be detected. If the downsampled version of the LIDAR is considered (Figure 4.11b), the small cell size provokes a steep reduction in the number of Gaussians, with the poles being barely detected, along with the ground. When the depth map is added however (Figure 4.11c), the poles are correctly detected and the ground is now sparser. It could be argued that the cell size is too low for the downsampled LIDAR scan. Increasing the cell size would however drown the details. Figure 4.12 illustrates this, with a cell size increased to 1.5 m: the downsampled LIDAR scan is used without the depth map addition. It can be seen that while the Gaussians are now less precise, the density is not significantly improved compared to the lower cell size.

The second scene is shown in Figure 4.13, it contains a bike leaning against a tree. When using the full LIDAR scan (Figure 4.14), the point cloud (Figure 4.14a) covers a reasonable area, although it is noticed that the projected depth map adds details to the bike and helps covering ground surfaces in between LIDAR beams. As a result, the NDT

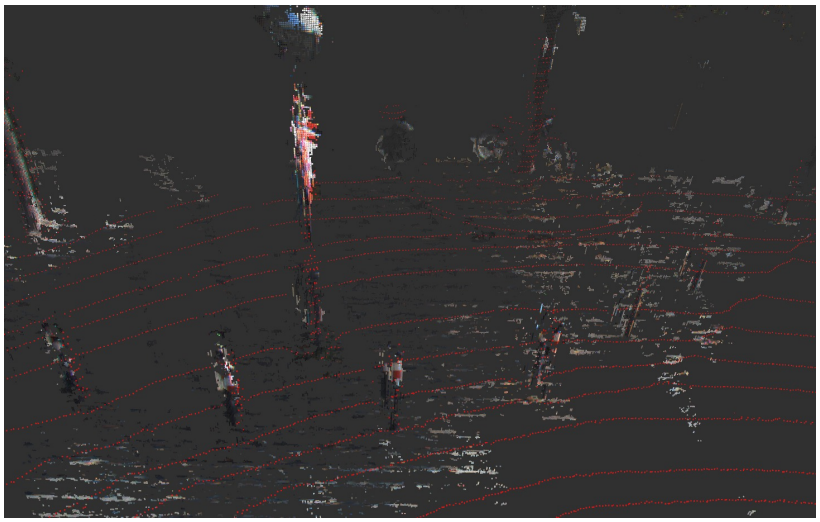
built from the LIDAR only (Figure 4.14b) is similar to the one using the LIDAR and the depth map (Figure 4.14c). The addition of the depth map allows more of the ground to be covered and some parts of the bike are better defined. The situation is very different for the downsampled LIDAR scan (Figure 4.15). In this case, the LIDAR does not cover well the bike nor the ground (Figure 4.15a). Since the LIDAR points are so sparse, the NDT built from the LIDAR scan (Figure 4.15b) only captures part of the bike and the tree and almost none of the floor. The addition of the depth map (Figure 4.15c) makes a significant improvement: the bike and the tree are now detected, along with some of the ground. Again, increasing the cell size does not really improve the NDT from the LIDAR only, as shown in Figure 4.16 where the Gaussians are too large to represent precisely the scene, while still missing a part of the bike.

To conclude this qualitative analysis, it can be seen that for LIDARs similar to the VLP-16, adding depth map data to the point cloud allows to perceive more in details small or medium objects. In the context of obstacle avoidance, this is an important advantage. This addition also fills gaps in between LIDAR beams, allowing a better coverage of planar surfaces such as the ground. The benefit of the depth map is reduced if the LIDAR coverage increases. If the distribution of the LIDAR points already covers the scene well, as in Figure 4.14, then there is little benefit to be gained by adding the depth map. If the coverage is sparser however, the depth map can help to fill the gaps and achieve a better NDT coverage, as in Figure 4.10.

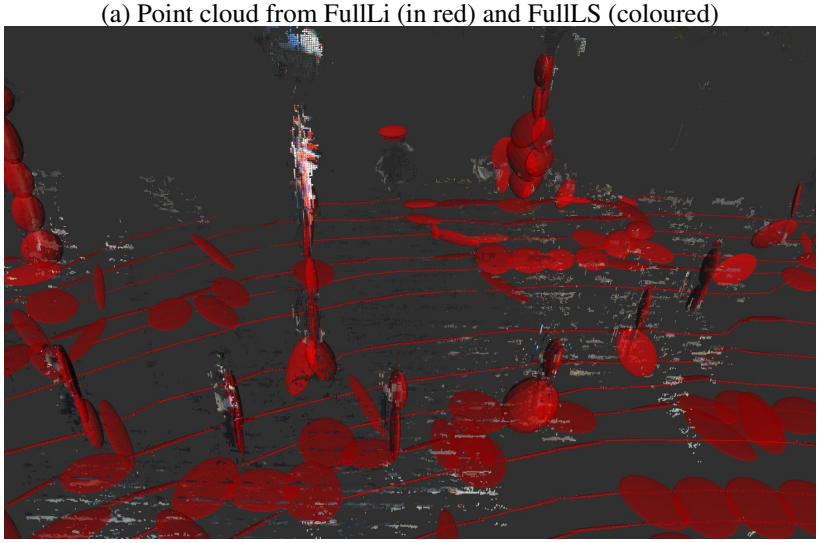
4.5.2 Precision: odometry error

After investigating the impacts of adding the depth map data to the LIDAR scan on the density of the resulting NDT, the effect on the precision of the map is examined in this section.

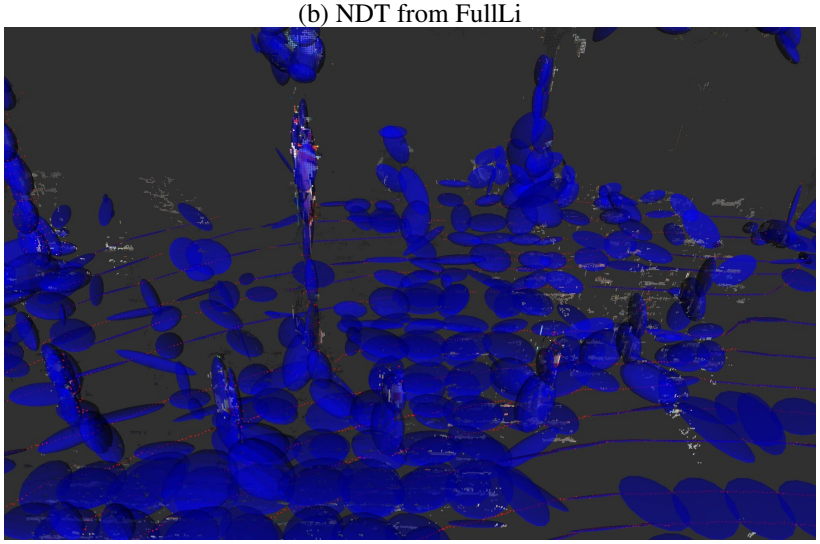
As explained in Section 3.5.2, the precision of the map can be evaluated through odometry, which is adopted again in this section. The dataset used is the KITTI odometry dataset [84]. Unless mentioned otherwise, the parameters used for the map are the one



(a) Point cloud from FullLi (in red) and FullLS (coloured)

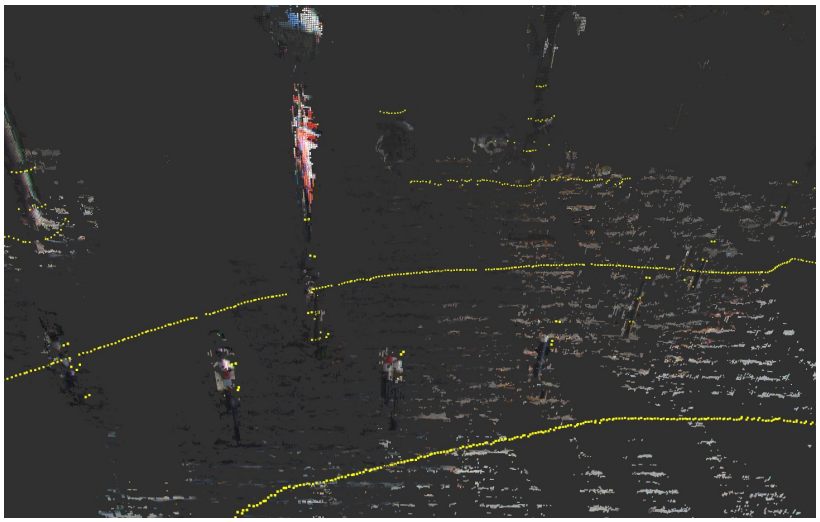


(b) NDT from FullLi

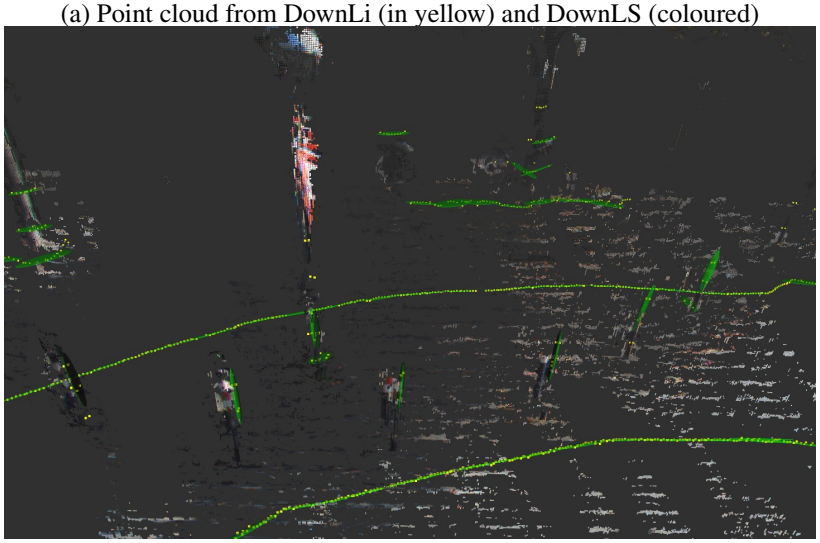


(c) NDT from FullLS

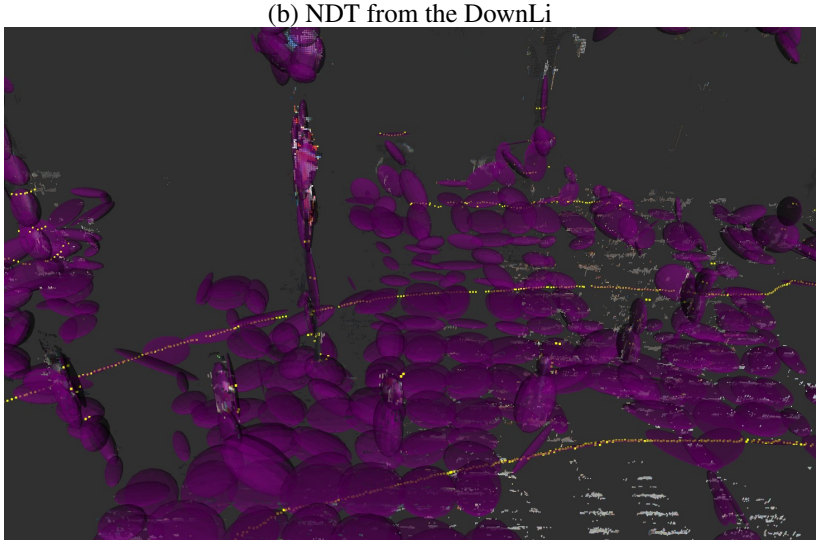
Figure 4.10: Point clouds and NDTs in scene 1 using the full LIDAR scan. First row shows the point clouds, second row the NDT built from the LIDAR data only, third row the NDT built from LIDAR and stereo data. Cell size is 0.5 m



(a) Point cloud from DownLi (in yellow) and DownLS (coloured)



(b) NDT from the DownLi



(c) NDT from the DownLS

Figure 4.11: Point clouds and NDTs in scene 1 using the downsampled LIDAR scan. First row shows the point clouds, second row the NDT built from the LIDAR data only, third row the NDT built from LIDAR and stereo data. Cell size is 0.5 m

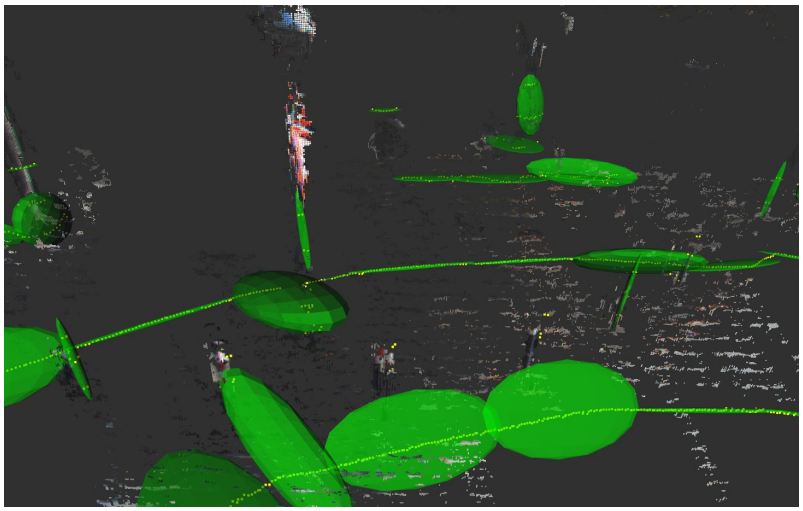


Figure 4.12: NDTs in scene 1 created from the downsampled LIDAR only, cell size of 1.5 m

Figure 4.13: Scene 2

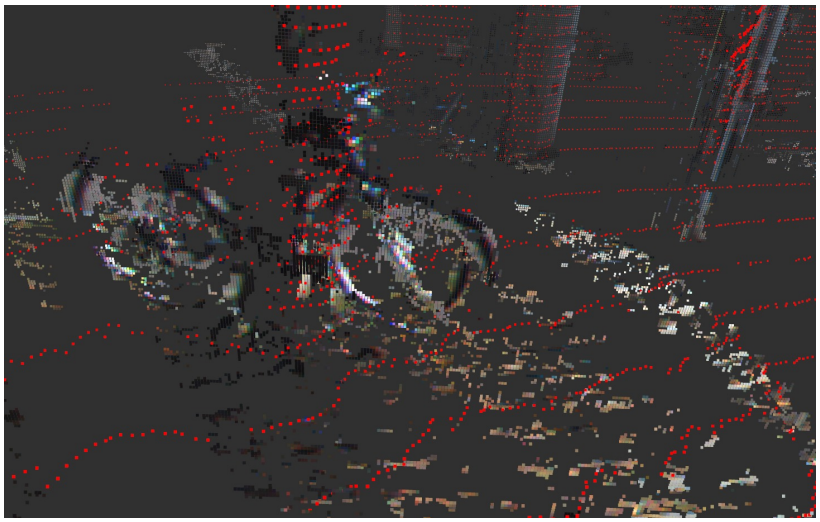
provided in Section 3.5.4.2. The error metrics are described in Section 3.5.4.

The odometry is evaluated for a point cloud created from two combinations of sensors: only stereo camera processed by the algorithm Tri_BF and a fusion of LIDAR and stereo data with the algorithm Tri_BF. In both cases, the depth map obtained from the algorithm Tri_BF is projected into a 3D point cloud. If the LIDAR is used, it is added to the point cloud obtained from Tri_BF. The resulting point cloud is fed to the mapping algorithm described in Section 3.3. Note that the case involving the raw point cloud of the LIDAR is examined in Section 3.5.4.2.

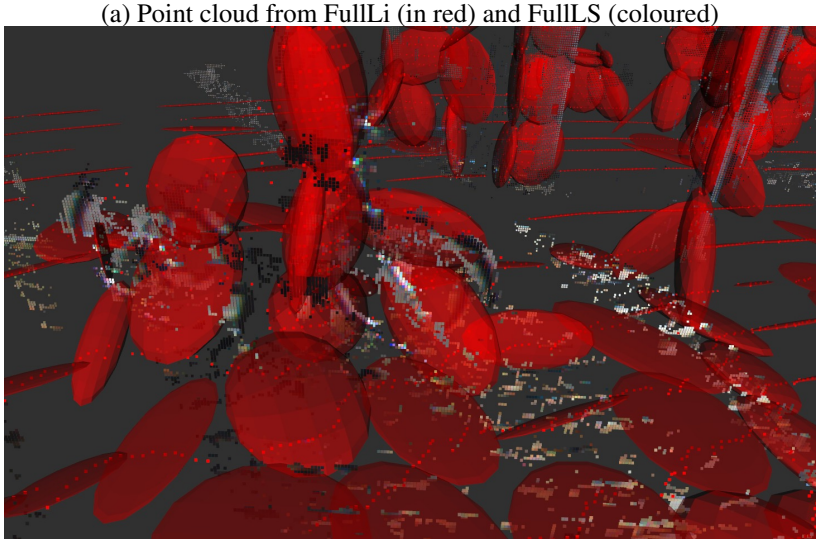
In order to quantify the change in precision, the odometry error is computed using two different variables: the depth cut-off values and the LIDAR type. A depth cut-off value of 60 m means that all points extracted from the Tri_BF algorithm farther from the sensor that 60 m are ignored. The LIDAR type is either full or downsampled, as defined in the previous section.

In Figure 4.17, the translation and rotation errors are plotted according to the depth cut-off values.

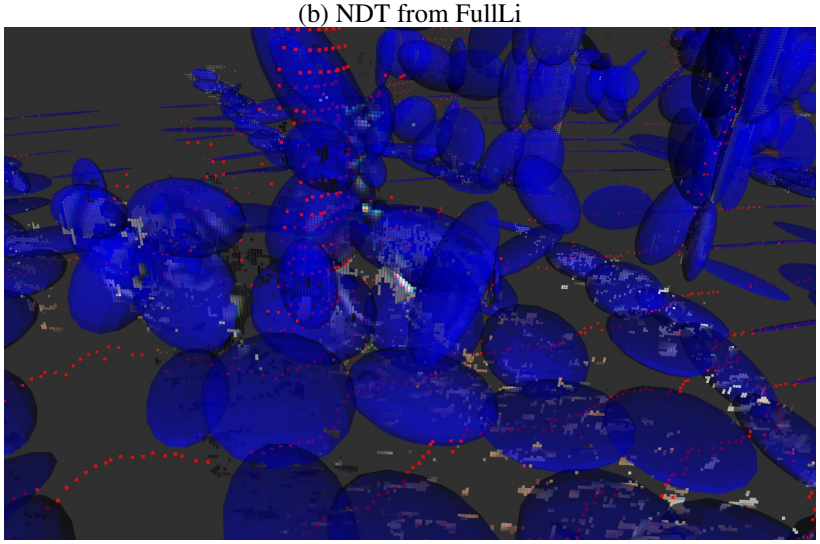
It is noticed that the point cloud built from the stereo alone is considerably less precise than the alternative. When the depth map is used, increasing the cut-off depth value results in a loss of precision, which is expected given that stereo cameras lose in precision when depth increases. Compared to the LIDAR alone, adding depth map data while limiting the depth at 15 m has a small effect on the odometry error. The errors, both in translation and rotation, slightly increase when the full LIDAR scan is used. However, the error decreases when the LIDAR scan is downsampled to a sensor compatible with a UAV. This result makes sense given the metric: odometry computation favours a small number of precise markers (in our case, Gaussian distributions). Thus, adding more points potentially less precise from a depth map might decrease the precision of the odometry if the LIDAR alone would be enough to compute the odometry. However, if the density of the LIDAR decreases, the number of Gaussian distribution coming from the LIDAR alone might drop sharply, as shown in Section 4.5.1. This might prevent the odometry to be computed



(a) Point cloud from FullLi (in red) and FullLS (coloured)

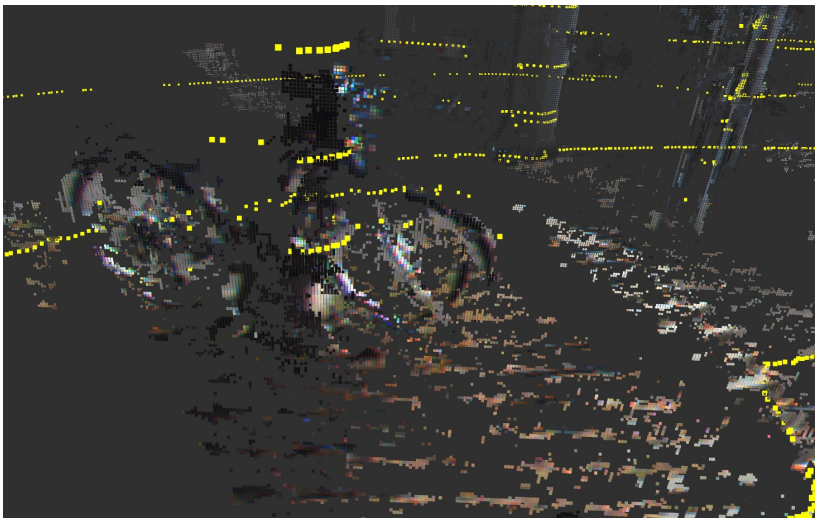


(b) NDT from FullLi

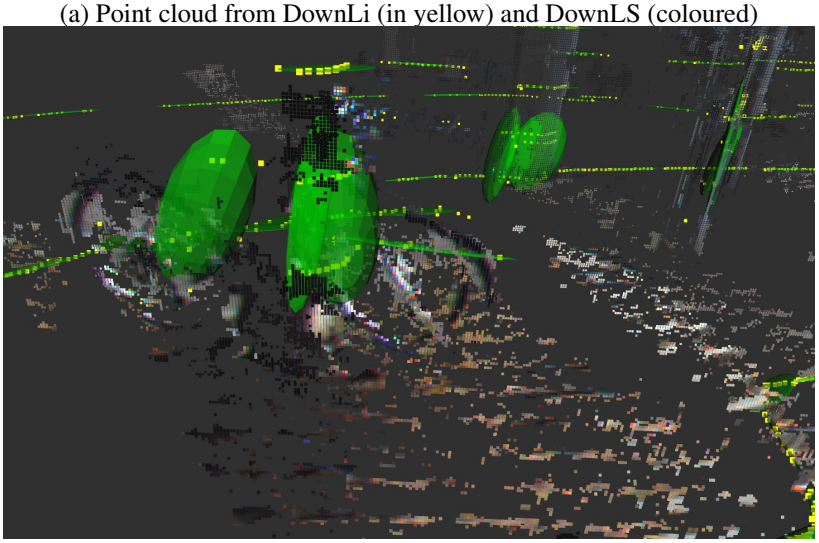


(c) NDT from FullLS

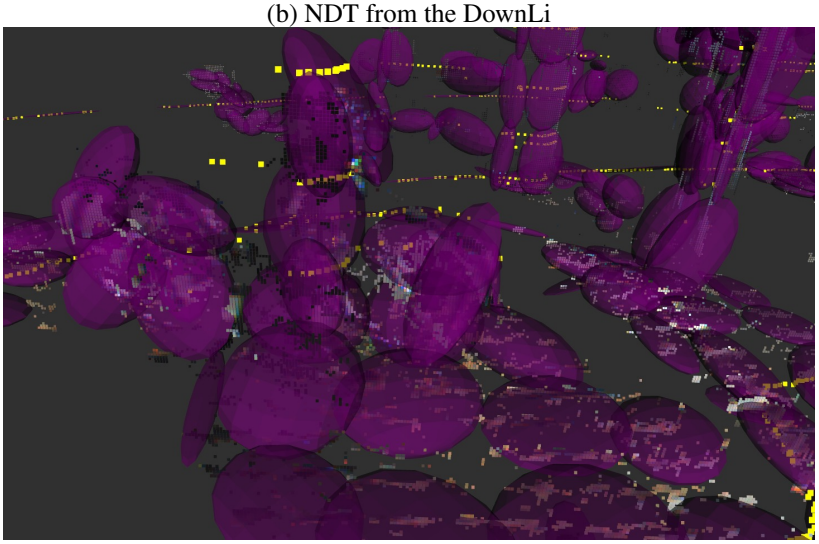
Figure 4.14: Point clouds and NDTs in scene 2 using the full LIDAR scan. First row shows the point clouds, second row the NDT built from the LIDAR data only, third row the NDT built from LIDAR and stereo data. Cell size is 0.5 m



(a) Point cloud from DownLi (in yellow) and DownLS (coloured)



(b) NDT from the DownLi



(c) NDT from the DownLS

Figure 4.15: Point clouds and NDTs in scene 2 using the downsampled LIDAR scan. First row shows the point clouds, second row the NDT built from the LIDAR data only, third row the NDT built from LIDAR and stereo data. Cell size is 0.5 m

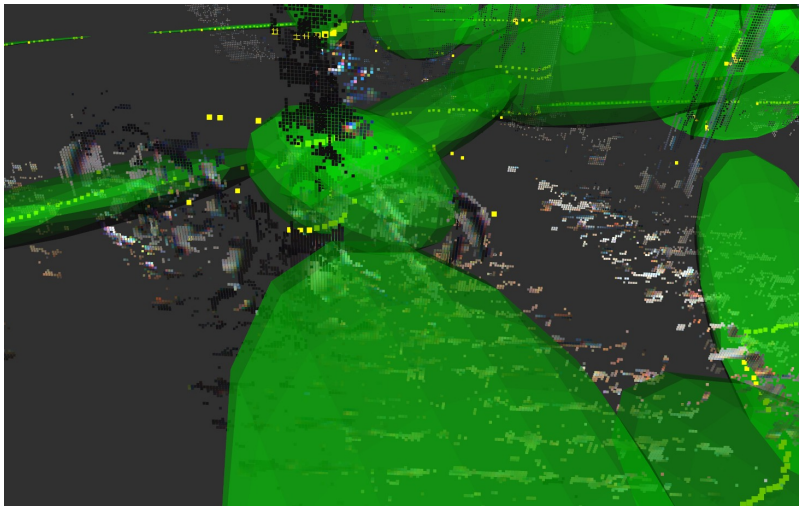


Figure 4.16: NDTs in scene 2 created from the downsampled LIDAR only, cell size of 1.5 m

Table 4.4: Runtime of the different methods on the KITTI dataset

Method	Runtime (ms)
Tri_BF, stereo only	462
Tri_BF, stereo and downsampled LIDAR	478
Tri_BF, stereo and full LIDAR	551
Raw full LIDAR only	187
Raw downsampled LIDAR only	84

precisely. Thus adding more Gaussian distribution from the depth map would contribute positively on the odometry computation, which is observed on Figure 4.17.

Taking into account the increase in density illustrated in Section 4.5.1 and the roughly unchanged or improved odometry shown in Figure 4.17, it is concluded that the addition of the depth map data from the method Tri_BF is recommended if the depth is limited to 15 m. Given the context of this thesis, navigation in closed spaces, this depth limitation is not restrictive, while the gain in density is important if small obstacles are present.

The runtime for one iteration of the different methods is illustrated in Table 4.4 on an Intel Core i7 6700 processor. For methods involving stereo data, the depth cut-off distance is 15 m. Adding the stereo data increased the runtime by approximately 400 ms, which is coherent with the runtime of the depth map computation without post processing.

In order to illustrate the difference in trajectory resulting from the addition of the depth

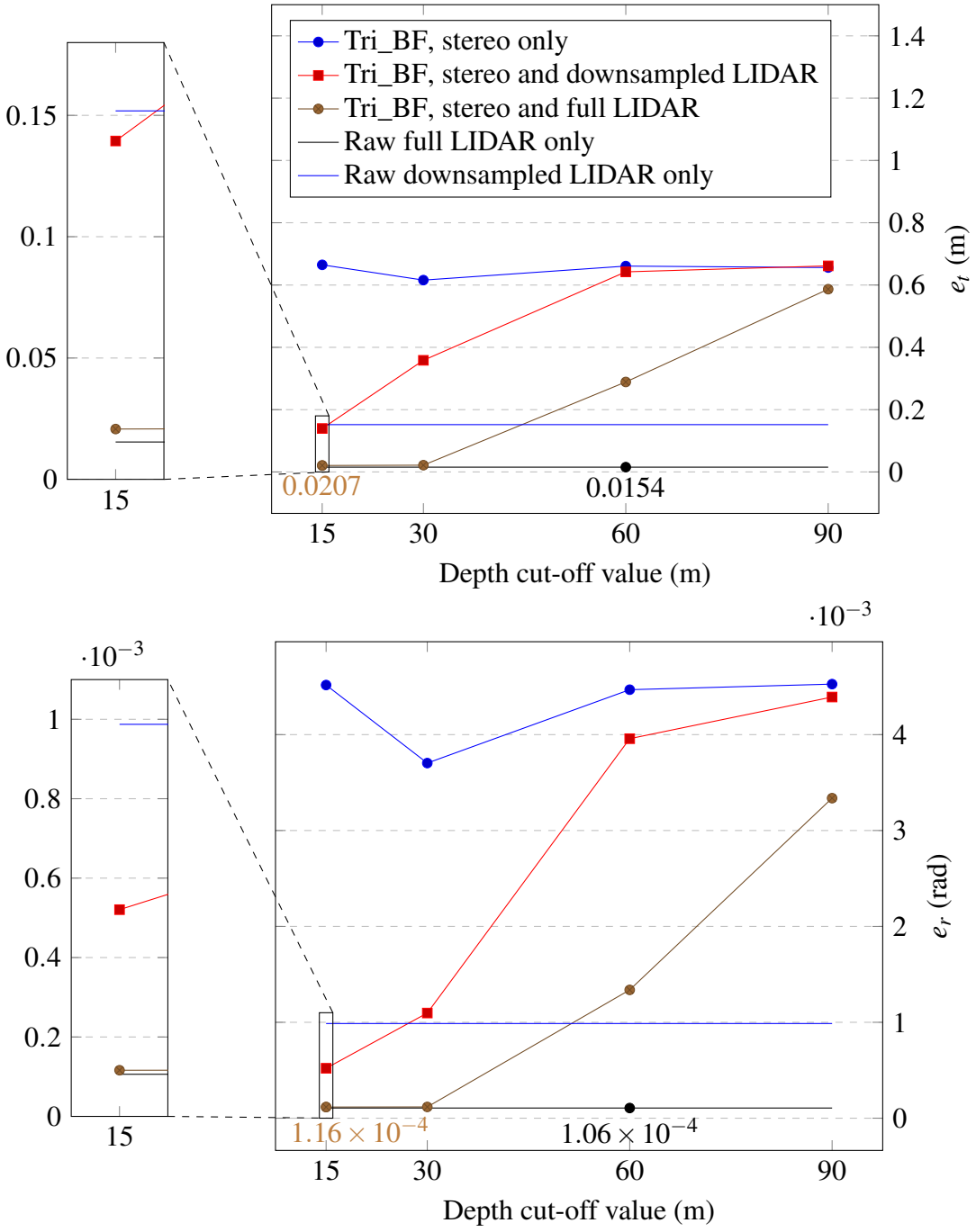
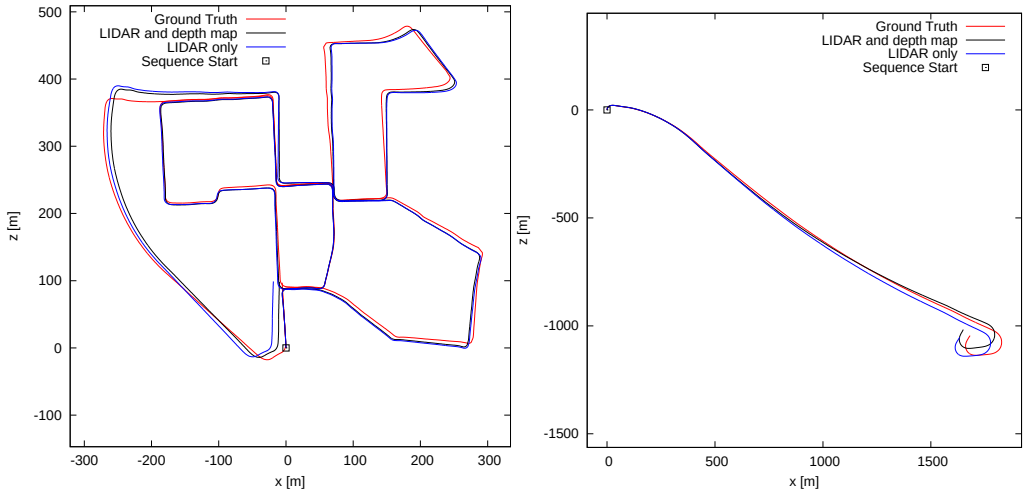


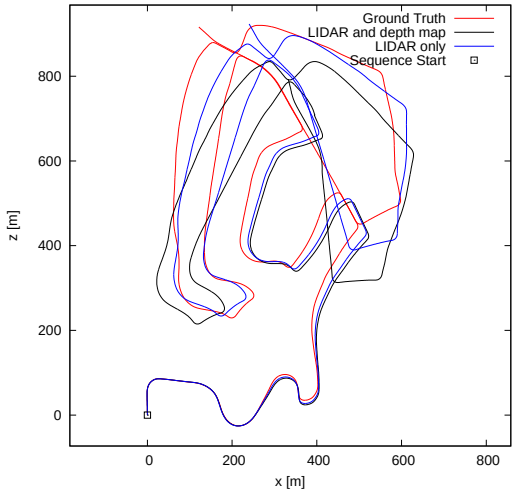
Figure 4.17: KITTI error depending on the depth cut-off distance

map, the trajectory obtained from the full LIDAR alone is plotted along with the trajectory from the full LIDAR and the depth map data with a cut-off depth at 15 m in Figure 4.18. As suggested by the small errors for both methods, the trajectories are very similar.

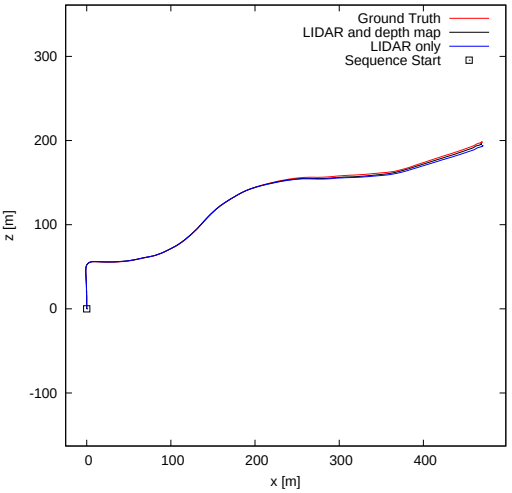


(a) Sequence 0

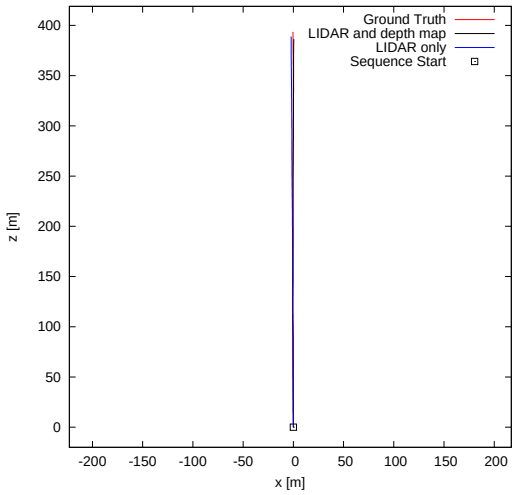
(b) Sequence 1



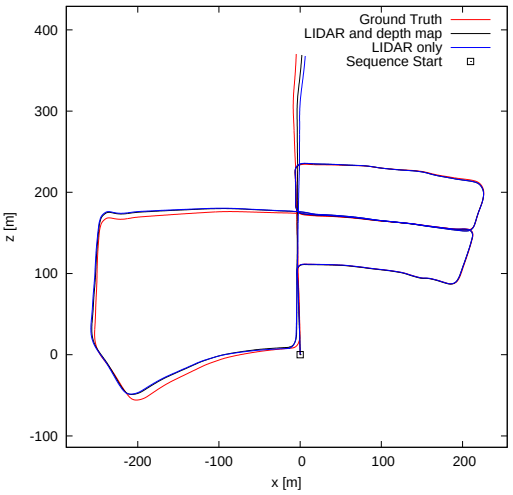
(c) Sequence 2



(d) Sequence 3

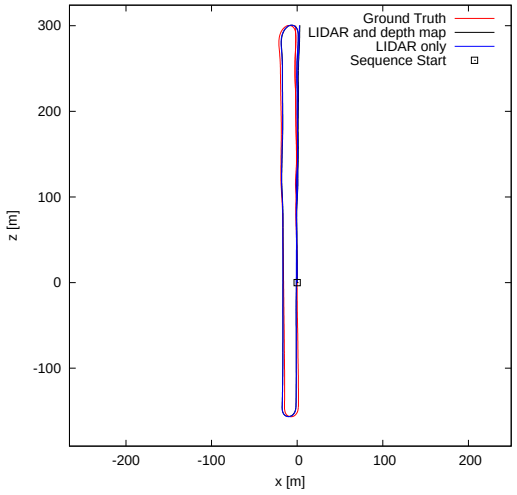


(e) Sequence 4

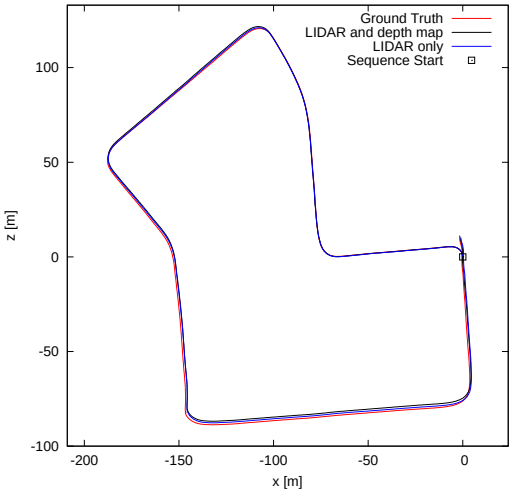


(f) Sequence 5

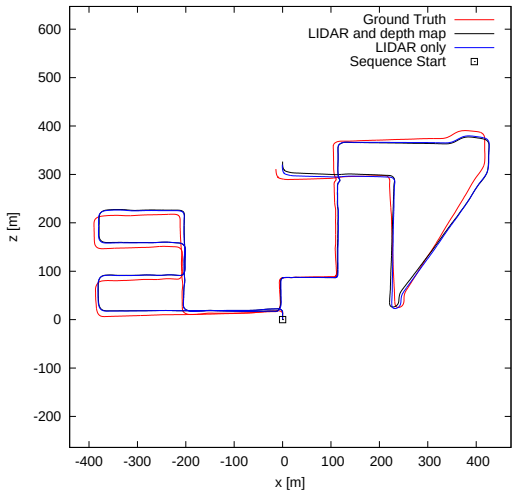
Figure 4.18: Trajectory on the KITTI dataset with the fusion of depth map and LIDAR data



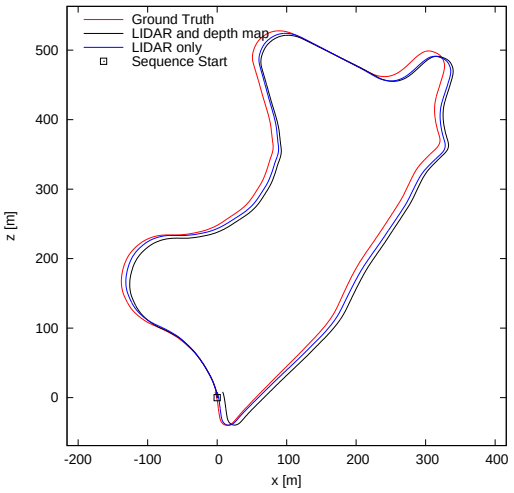
(g) Sequence 6



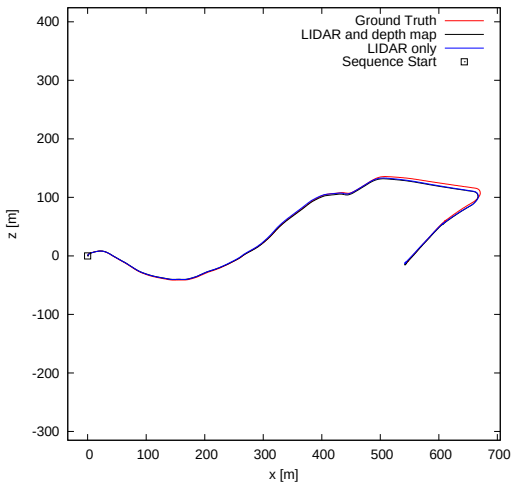
(h) Sequence 7



(i) Sequence 8



(j) Sequence 9



(k) Sequence 10

Figure 4.18: Trajectory on the KITTI dataset with the fusion of depth map and LIDAR data

4.6 Conclusion

In order to increase the density of the point cloud used for mapping, a new technique to compute dense depth map in real time by fusing stereo and LIDAR data has been presented. The proposed method, Tri_BF, is based on a modified bilateral filter in a Bayesian framework. An analysis of the quality of the generated depth map on the KITTI dataset is carried out. Tri_BF is compared with other methods based on fusion, or extended to work with fusion and is shown to be competitive with those methods.

The influence of the depth map on the mapping process is investigated. It is shown that, in two specific scenes, the depth map resulting from the proposed algorithm allows the detection of more obstacles when using a LIDAR compatible with small UAVs in terms of weight, size and power consumption. The influence of the depth map on the odometry is shown to be negligible under the right cut-off distance for the depth map when the full LIDAR is used. Should a LIDAR suitable for UAVs be used, the depth map is shown to improve the odometry. The runtime increase is however considerable, although a GPU implementation could alleviate this problem.

Overall, the addition of the depth map data computed from Tri_BF has a positive impact on the mapping process. The mapping part of the flight assistance scheme is now complete, in the next chapter, the complete system architecture is described.

Chapter 5

OAST: obstacle avoidance system for teleoperation

Contents

5.1 Introduction

This chapter presents a flight assistance system, OAST (Obstacle Avoidance System for Teleoperation), whose main role is UAV obstacle avoidance and assistance to trajectory generation. A new obstacle avoidance strategy which relies on the map created in previous chapters is proposed. The integration of the flight assistance system components is discussed. OAST is tested in computerised experiments that validate the design.

5.1.1 Different types of obstacle avoidance

UAV tasks can be conducted in two manners: as autonomous missions, where the UAV is not under direct human supervision or as teleoperated missions, where the UAV is remotely controlled by a human operator.

In the case of autonomous missions, path planning has to be performed. Modern path planners are often divided in two categories: a global path planner computing the overall trajectory, while a local path planner computes a more refined path locally [113, 114]. The reason for the existence of the local path planner is that the exact environment is often not known, so there is a need for a refinement of the path according to the newly detected obstacles around the robot.

The scope of this work is intelligent UAV teleoperation: in this context, the operator is responsible for the global planning, eliminating the need for a global planner. For some teleoperation scenarios such as inspection or search and rescue, the operator might adopt an exploratory approach, changing the desired position of the UAV as required. A global path planner might add a significant computational overhead that is unnecessary: depending on the type of controller used, the frequency of user inputs is expected to be relatively high, probably between 10 Hz and 100 Hz. With such a high frequency, the robot would have a limited time to follow the trajectory before the next command comes in. This means that all the computational resources used to compute the trajectory might be wasted if the new input is different enough from the previous one. While some path planning algorithms can cope with a change of starting point (such as the D* family of algorithms [115, 116]), they generally assume a fixed end point.

The objective of the desired system is then to help the operator to avoid obstacles, while keeping a liberty of movement. To achieve this objective, it is proposed to filter the input received from the operator. A position is proposed by the operator through the joystick. If this position and the path leading to it are safe considering the 3D map available, the UAV progresses towards it. Otherwise, the UAV stops early to avoid the collision while still going as close as possible to the obstacle. In the later case, it is

proposed to use the map to try to find a new desired position close to the intent of the operator.

It is important to highlight the differences with the obstacle avoidance scheme designed in Chapter 2. In that chapter, the haptic feedback was used to stir the joystick in a direction opposing an obstacle. As a result, the UAV itself would go away from the obstacle. In this chapter, a safe position is computed directly from the 3D map. The main advantage of this new approach is that collisions are now expected to be less likely, since a safe position is imposed. A second advantage is that the obstacle avoidance algorithm is now independent from the joystick mechanical characteristics and human pilot, thus tuning is simplified. A third advantage is that the manoeuvrability is increased: in Chapter 2, the aim of the algorithm was to prevent collision, meaning that the UAV would stop before a collision. In this chapter, the role of the algorithm goes further. It must prevent collisions while trying to achieve the original position goal of the operator. The drawback is that the human does not have complete control anymore: some unsafe positions are now impossible to reach. This drawback can be mitigated by authorising the operator to deactivate the safety scheme. As a result, the haptic feedback has now a purely informative role to the operator as it is not used to actively avoid obstacles. The haptic feedback can facilitate the interface between the user and the system.

5.1.2 Flight assistance system components

The system architecture is composed of five parts, which will be described in more details in their respective sections. The role of each is mentioned below to provide a better overview of the complete system. A special care is given to the obstacle avoidance module which is a key development in this chapter. The whole system is illustrated in Figure 5.1.

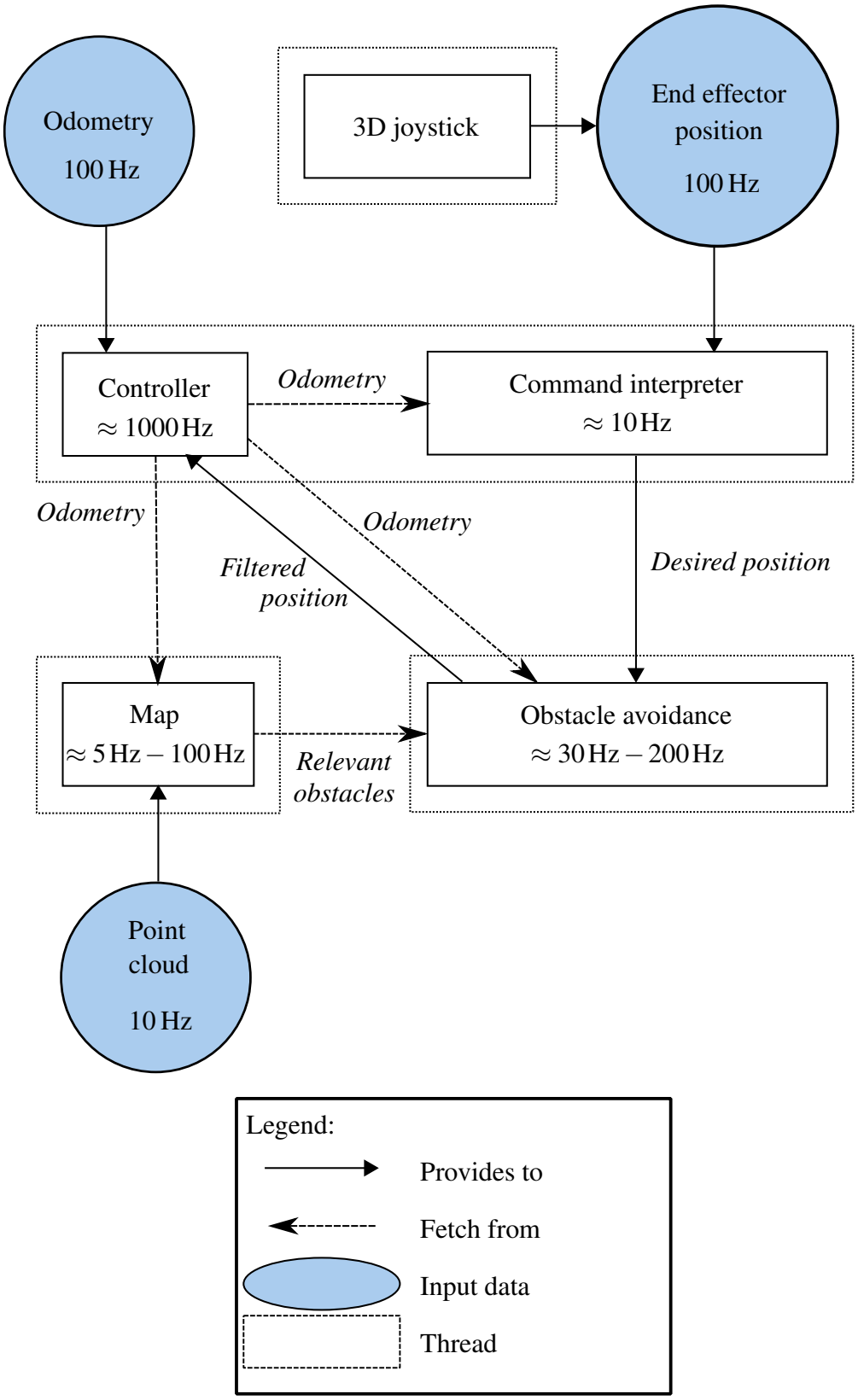


Figure 5.1: Complete system for UAV obstacle avoidance

5.1.2.1 The map:

The map is described in Chapter 3, the specific implementation is detailed in Section 3.3.1 with a density optimisation proposed in Chapter 4. This 3D map takes a point cloud as input. The output can be described by a list of Gaussian distributions describing the obstacles around the UAV. A typical update rate for the input point cloud is 10 Hz. The map update rate depends mainly on the map resolution, as shown in Section 3.5.2. For a map resolution of 2.2 m and width of 240 m, the map update rate is 4.7 Hz with odometry computation and 100 Hz without computing the odometry (on the Ford dataset, see Section 3.5.2 for details). If the odometry is not computed, it has to be provided by an external source positioning sensor.

5.1.2.2 The 3D haptic controller:

The haptic controller is operating in impedance mode, meaning that the input from the haptic controller is the position of the end effector, which is sent to the UAV. The haptic controller can then receive back a feedback from the system. The new haptic feedback is discussed in Section 5.5. The haptic controller sends commands to the UAV at a rate of 100 Hz.

5.1.2.3 The command interpreter:

The command interpreter transforms the joystick input into an actual objective for the UAV. In this work, a position objective is considered. The command interpreter is discussed in Section 5.2.

5.1.2.4 The UAV controller:

Given an objective, the controller computes the required velocity of each propeller to reach the goal. The controller that is used here accepts both velocity and position commands [24]. While this controller was implemented in the RotorS package [40] that was

used to simulate the UAV in Chapter 2, it has been reimplemented in the proposed system for integration purposes. The controller runs at 1000 Hz in this implementation.

5.1.2.5 The obstacle avoidance and trajectory assistance module:

This module takes as an input an objective, i.e. the output of the command interpreter. It then retrieves the whole map (or parts of the map) to amend the objective to keep the UAV safe while respecting the operator's input. It finally transmits the filtered objective to the UAV controller. This module, which is the core of this chapter, is described in Section 5.3.

5.1.2.6 Implementation notes

The whole system is separated in two programs. The first one is the 3D haptic controller, which runs on the computer of the operator and sends data to the UAV. The second program is supposed to run on the UAV. It is composed of the command interpreter, the UAV controller, the map and the obstacle avoidance module. The communication between those two programs and the sensors is done using ROS [38]. The simulator used is Gazebo [39], along with the RotorS [40] package for the physical model of the UAV. Our system receives two external types of data: the point clouds from the LIDAR (received by the map) and the odometry (received by the UAV controller).

As shown in Chapter 3, the map can generate odometry data. The accuracy of this odometry is reported in Section 3.5.2. Since this chapter aims to evaluate the obstacle avoidance scheme independently, it is desirable to decouple the odometry generation from the proposed architecture. For this reason, an external source of odometry is used as a ground truth in the following developments. For more details about methods to compute odometry for UAVs, please refer to Section 3.1.3.

The program that runs on the UAV is separated in four threads: one for handling the inputs (point cloud, odometry and haptic controller commands), one for the map, one shared by the command interpreter and the controller and one for the obstacle avoidance

module. Note that the program is implemented so that each thread runs in a loop using the latest data available, no thread waits for the data. For instance, the controller does not wait for a new command or a new odometry data: when a new command or odometry data arrives, it is stored in memory and the controller always use the latest data. This is interesting because as seen above, the sensors that are used have different update rate. Thus it is not desirable for the obstacle avoidance module to wait for the a new point cloud to arrive before generating a new safe objective. Indeed the point cloud are updated at 10 Hz, but the odometry is updated at 100 Hz and new commands arrive at 100 Hz. This means it is possible to reuse the map while updating the odometry or commands to generate a new safe objective position, which is the way the code was written.

In the following sections, the individual parts of the system are described in more detail.

5.2 Command interpreter module

5.2.1 An overview of the former approach in the current context: a wind up problem

In this new proposed system, the command from the user is filtered and the haptic feedback is purely informative. This means that when the UAV is blocked by the program because there is an obstacle in the way, the user can continue freely to push in the direction of the obstacle. The former scheme introduced in Chapter 2 to transform the end effector displacement into a position objective is described in Section 2.2.1. This method suffers from a problem in the context of the proposed obstacle avoidance method: position objective can drift far from the current UAV position (since the UAV does not move toward the objective, it is blocked by the obstacle avoidance module). The consequence is a latency when the user finally changes direction and possibly sudden movements.

In order to reduce this wind up effect, it is possible to limit the distance between the UAV and the position goal to a maximum value $L_{\max}$. The transformation from a

displacement of the end effector $\mathbf{p_j}(t)$ into a position objective $\mathbf{x_r}(t)$ is described in equation (2.3). The objective is to modify this equation to ensure that, given a position of the UAV $\mathbf{x_{UAV}}(t)$, the following equality holds for all times t:

$$\|\mathbf{x_{UAV}}(t) - \mathbf{x_r}(t)\| \leq L_{\max}. \tag{5.1}$$

Algorithm 5.1 proposes a solution to this issue, its derivation is shown below. An approach allowing an easier control by the user is proposed in Section 5.2.2.

Let us remind equation (2.3), originally from Omari et al. [36]:

$$\mathbf{\Psi}(t) = \mathbf{K_v}\mathbf{p_j}(t)\max\left(0, 1 - \frac{r^*}{\|\mathbf{p_j}(t)\|}\right), \tag{5.2}$$

$$\mathbf{x_c}(t) = \int_0^t \mathbf{\Psi}(s)\mathrm{ds}, \tag{5.3}$$

$$\mathbf{x_r}(t) = \mathbf{K_p}\mathbf{p_j}(t)\min\left(1, \frac{r^*}{\|\mathbf{p_j}(t)\|}\right) + \mathbf{x_c}(t). \tag{5.4}$$

The positive definite matrices $\mathbf{K_v}$ and $\mathbf{K_p}$ serve scaling purposes and r^* is the radius of the virtual sphere in the haptic controller workspace. For more details, see Section 2.2.1.

In order to have a better view of the necessary conditions to bind $\mathbf{x_r}$, $\mathbf{x_c}$ is rewritten:

$$\mathbf{x_c}(t) = \int_0^t \mathbf{\Psi}(s)\mathrm{ds}, \tag{5.5}$$

$$= \mathbf{x_c}(t-1) + \int_{t-1}^t \mathbf{\Psi}(s)\mathrm{ds}. \tag{5.6}$$

Let us consider a discrete representation of time, with a constant time step that is absorbed into $\mathbf{K_v}$. Then from equation (5.6):

$$\mathbf{x_c}(t) = \mathbf{x_c}(t-1) + \frac{\mathbf{\Psi}(t-1) + \mathbf{\Psi}(t)}{2}. \tag{5.7}$$

Assuming that $\mathbf{\Psi}(t \leq 0) = \mathbf{0}$, a recurrence reasoning leads to:

$$\mathbf{x_c}(t) = \sum_{i=0}^{t-1} \mathbf{\Psi}(i) + \frac{\mathbf{\Psi}(t)}{2}. \tag{5.8}$$

We define: $\mathbf{S}(t) = \sum_{i=0}^{t} \mathbf{\Psi}(i)$.

Then equation (5.4) can be rewritten as:

$$\mathbf{x_r}(t) = \begin{cases} \mathbf{K_p p_j}(t) + \mathbf{S}(t-1), & \text{if } \|\mathbf{p_j}(t)\| < r^*, \\ \left(\frac{r^*}{\|\mathbf{p_j}(t)\|} \left(\mathbf{K_p} - \frac{\mathbf{K_v}}{2} \right) + \frac{\mathbf{K_v}}{2} \right) \mathbf{p_j}(t) + \mathbf{S}(t-1), & \text{if } \|\mathbf{p_j}(t)\| \geq r^*. \end{cases} \tag{5.9}$$

From equation (5.9), there are two components: $\mathbf{S}$ (related to $\mathbf{x_c}$ by $\mathbf{x_c}(t) = \mathbf{S}(t-1) + \mathbf{\Psi}(t)/2$) and a term proportional to $\mathbf{p_j}(t)$. To impose the limitation of equation (5.1), it is necessary to limit those two terms. Let us define a scaled end effector displacement $\mathbf{p_s}(t)$ as:

$$\mathbf{p_s}(t) = \begin{cases} \mathbf{K_p p_j}(t), & \text{if } \|\mathbf{p_j}(t)\| < r^*, \\ \left(\frac{r^*}{\|\mathbf{p_j}(t)\|} \left(\mathbf{K_p} - \frac{\mathbf{K_v}}{2} \right) + \frac{\mathbf{K_v}}{2} \right) \mathbf{p_j}(t), & \text{if } \|\mathbf{p_j}(t)\| \geq r^*. \end{cases} \tag{5.10}$$

This means that $\mathbf{x_r}$ is now:

$$\mathbf{x_r}(t) = \mathbf{p_s}(t) + \mathbf{S}(t-1). \tag{5.11}$$

The first proposed approach is to limit the growth of $\mathbf{S}$, then scale $\mathbf{p_s}$ if needed. Let $\mathbf{x_s}(t)$ be the redefined position goal:

$$\mathbf{x_s}(t) = \alpha \mathbf{p_s}(t) + \mathbf{S}(t-1), \tag{5.12}$$

with $\alpha \in [0,1]$ so that equation (5.1) holds. This means that if $\|\mathbf{x_s}(t) - \mathbf{x_{UAV}}(t)\| \leq L_{\max}$ with $\alpha = 1$, then $\alpha = 1$ is kept.

Otherwise, to determine alpha, the following polynomial has to be solved:

$$\|\mathbf{x_s}(t) - \mathbf{x}_{\text{UAV}}(t)\|^2 - L_{\max}^2 = 0, \tag{5.13}$$

$$\iff \alpha^2 \|\mathbf{p_s}(t)\|^2 - 2\alpha \mathbf{p_s}(t)\left(\mathbf{S}(t-1) - \mathbf{x}_{\text{UAV}}(t)\right) + \|\mathbf{S}(t-1) - \mathbf{x_{UAV}}\|^2 - L_{\max}^2 = 0. \tag{5.14}$$

If there are two roots of same sign to this polynomial, only one root, or no root, it means that the constraint set by equation (5.1) is already violated. Then $\alpha = 0$ is chosen. If there are two roots of different signs, then the positive one is kept. Note that enforcing the inequality $\|\mathbf{S}(t-1) - \mathbf{x_{UAV}}(t)\| \leq L_{\max}$ guarantees the existence of a solution $\mathbf{x_s}$ to equation (5.13).

The complete algorithm to determine the position $\mathbf{x_r}(t)$ when a new end effector position comes in is summarised by Algorithm 5.1.

After testing, it was found that this algorithm works well in the absence of obstacles. However, it is difficult to manoeuvre the UAV near obstacles. While the anti wind up strategy described above reduces the latency in the controls, it does not make it disappear. For this reason, the command interpreter uses a direct mapping instead, that is described in the next section.

5.2.2 A better strategy: direct proportional mapping

5.2.2.1 Direct mapping

As mentioned in Section 2.2.1, there are different ways to achieve unlimited movement from a limited workspace of the end effector. A popular technique is velocity mapping: the displacement of the end effector sets a velocity target for the robot, or equivalently,

Algorithm 5.1: Algorithm to determine the position goal of the drone from an input of the haptic controller at time step t

Input : Position of the end effector of the controller $\mathbf{p_j}(t)$, a maximum threshold for the distance of the UAV to the position goal $L_{\max}$, the position of the UAV $\mathbf{x_{UAV}}(t)$

Output: A position goal $\mathbf{x_r}(t)$ for the UAV, $\mathbf{S}(t)$ is updated

Compute $\mathbf{\Psi}(t) = \mathbf{K_v}\mathbf{p_j}(t)\max\left(0, 1 - \frac{r^*}{\|\mathbf{p_j}(t)\|}\right)$;
if $\|\mathbf{S}(t-1) - \mathbf{x_{UAV}}(t)\| > L_{\max}$ **then**
  $\mathbf{S}(t-1) = \mathbf{x_{UAV}}(t) + L_{\max}\frac{\mathbf{S}(t-1)-\mathbf{x_{UAV}}(t)}{\|\mathbf{S}(t-1)-\mathbf{x_{UAV}}(t)\|}$;
end
if $\|\mathbf{p_j}(t)\| < r^*$ **then**
  $\mathbf{p_s}(t) = K_p\mathbf{p_j}(t)$;
else
  $\mathbf{p_s}(t) = \left(\frac{r^*}{\|\mathbf{p_j}(t)\|}\left(\mathbf{K_p} - \frac{\mathbf{K_v}}{2}\right) + \frac{\mathbf{K_v}}{2}\right)\mathbf{p_j}(t)$;
end
Compute $\mathbf{x_r}(t) = \mathbf{p_s}(t) + \mathbf{S}(t-1)$;
if $\|\mathbf{x_r}(t) - x_{\mathrm{UAV}}(t)\| > L_{\max}$ **then**
  Solve $\alpha^2\|\mathbf{p_s}(t)\|^2 - 2\alpha\mathbf{p_s}(t)\left(\mathbf{S}(t-1) - \mathbf{x_{UAV}}(t)\right) + \|\mathbf{S}(t-1) - \mathbf{x}_{\mathrm{UAV}}(t)\|^2 - L_{\max}^2 = 0$ for α;
  if *there are two real roots, one negative and one positive* **then**
    Let the positive root be α^*;
    $\mathbf{x_r}(t) = \alpha^*\mathbf{p_s}(t) + \mathbf{S}(t-1)$;
  else
    $\mathbf{x_r}(t) = \mathbf{S}(t-1)$;
  end
end
if $\|\mathbf{S}(t-1) + \mathbf{\Psi}(t) - \mathbf{x_{UAV}}(t)\| \leq L_{\max}$ **then**
  $\mathbf{S}(t) = \mathbf{S}(t-1) + \mathbf{\Psi}(t)$;
else
  $\mathbf{S}(t) = \mathbf{S}(t-1)$;
end
return $\mathbf{x_r}(t)$;

the displacement of the robot is the integral of the displacement of the end effector. The scheme described in Section 2.2.1 and modified above is using velocity mapping, but applied to the target position of the robot and not the robot itself. The advantage is a finer control over small displacement, but it implies a latency. When the target position is forward, this position has to "travel" behind the UAV before the robot can actually move back.

The other solution is to use a direct velocity mapping. This makes small displacement more difficult to achieve, but the latency disappears. During testing, it was found easier to manoeuvre near obstacles with a direct velocity control. However, since a desirable feature of UAVs is their capacity to hover, the scheme is modified to allow hovering.

For a given displacement of the end effector $\mathbf{p_j}(t)$, a standard velocity mapping would be, for a velocity of the UAV $\dot{\mathbf{x}}_\mathbf{r}(t)$:

$$\dot{\mathbf{x}}_\mathbf{r}(t) = k\mathbf{p_j}(t), \tag{5.15}$$

with k a scaling factor.

5.2.2.2 Hovering addition

The scheme used in this chapter for position control of the UAV is the following. First, the displacement of the end effector $\mathbf{p_j}(t) = [p_j^x(t), p_j^y(t), p_j^z(t)]$ is normalized per component, so that each component is between -1 and 1. This allows the proposed algorithm to be independent from the actual size of the workspace. This also means that the workspace is approximated by a rectangular cuboid (this is done because the workspace of a 3D haptic controller is often not spherical). Let the function h_j be:

$$h_j(x, r^*) = \begin{cases} 0, & \text{if } |x| < r^*, \\ x - sign(x)r^*, & \text{if } |x| \geq r^*, \end{cases} \tag{5.16}$$

with $x \in \mathbb{R}$, $r^* \in]0,1]$ and *sign* a function returning the sign of its argument. Then,

assuming that the desired yaw is θ_y, the desired position is given by:

$$\mathbf{x_r}(t) = \mathbf{x_{UAV}}(t) + \begin{bmatrix} \cos(\theta_y) & -\sin(\theta_y) & 0 \\ \sin(\theta_y) & \cos(\theta_y) & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{K}_s \begin{bmatrix} h_j\left(p_j^x(t), r^*\right) \\ h_j\left(p_j^y(t), r^*\right) \\ h_j\left(p_j^z(t), r^*\right) \end{bmatrix}, \tag{5.17}$$

where $\mathbf{x_r}(t)$ is the position goal of the UAV, $\mathbf{x_{UAV}}(t)$ is the current position of the UAV, $\mathbf{K}_s$ is a definite positive scaling matrix and r^* is a real scalar.

If the component of the displacement of the end effector is less than r^*, then the desired displacement is zero for this component (i.e. the position goal is the current position of the UAV for this component). This means that hovering can be maintained. From r^*, the norm of the desired displacement grows linearly with the norm of the displacement of the end effector. Note that this way, equation (5.1) is valid, with $L_{\max} = \sqrt{3}\lambda_s(1-r^*)$ and λ_s being the largest eigenvalue of $\mathbf{K}_s$.

Unlimited displacement is ensured by setting the position goal relative to the position of the UAV. A possible drawback of this method is that the UAV can drift when the requested displacement is small (an integral scheme to cancel the steady state error is less efficient since the position goal changes with the position of the UAV).

5.3 Obstacle avoidance module

The objective of the obstacle avoidance module is to prevent collisions of the UAV with the surrounding obstacles and to facilitate the operation of the UAV. First, the literature related to this work is presented, then the proposed approach for OAST is described.

5.3.1 Related work

It is important to underline the difference between path planning and obstacle avoidance: path planning aims to find a continuous path between a start point and an end point to be

used by the robot. Obstacle avoidance is the task of reactively avoiding obstacles on the path of the robot. As such, obstacle avoidance can be seen as a part of path planning [114].

As mentioned in the introduction of this chapter, path planning is out of the scope of this work since it is not necessary for teleoperation. For the reader's interest, a recent overview of the different 3D path planning algorithms is available in Yang et al. [117]. The main limitation of path planning is the computational power needed. However, some algorithms have been proposed very recently to tackle the problem of 3D path planning in real time. Allen and Pavone [118] propose an algorithm split in two phases: an offline phase, where a Support Vector Machine classifier is used to shorten the online computation by approximating state neighbourhood. The optimal trajectory is obtained with a random sampling algorithm: kino-FMT*. Finally, the trajectory is smoothed. The runtime on a physical robot varies between 0.20 s and 0.33 s for 1000 sampled nodes. A second real time algorithm is proposed by Sanchez-Lopez et al. [119]. This method describes the environment with geometric primitives. The admissible space is randomly sampled and the A^* algorithm is used to generate the trajectory.

Obstacle avoidance is however the core component of the proposed flight assistance system. The topic of collision free navigation in cluttered environment is still an active research domain [120]. Below, some popular obstacle avoidance methods for ground robots at first, then UAVs are listed.

5.3.1.1 For ground vehicles

A popular category of methods for ground robots are potential fields methods, such as Virtual Force Field [121], where the robot is repelled away from obstacles based on their distance. An inherent weakness of those methods is the possibility for the robot to be stuck in local minima. An attempt to solve this problem of the Virtual Force Field was made with the Vector Field Histogram method [122], where the space is divided in polar zones. A histogram of the density of obstacles per polar zone is then computed, then the zone of lowest density is chosen. The choice of the zone was further improved while

taking the dimension of the robot into account [123] and path planning capabilities were added [124]. The idea of using a polar histogram was used in other methods [125, 126].

Another popular method for ground robots is the Dynamic Window Approach [127], that searches in both the available physical space and the reachable velocity space of the robot for a solution. A drawback of this method is the increase in computational complexity when the search space grows.

A third category of methods is the elastic bands [128] which takes a starting path and refines it so that is is shorter and smoother.

5.3.1.2 For UAVs

The passage from ground robots to UAVs involves higher computational requirements (to account for the additional space dimension) and different dynamics [120]. This makes the conversion of those methods to 3D sometimes difficult. The Vector Polar Histogram method was used with UAVs [129, 130] but the obstacle detection was limited to 2D. The Dynamic Window Approach is starting to see some uses for flying robots [131, 132].

Teleoperation is the area of interest of this thesis, thus existing work related to obstacle avoidance for teleoperated UAVs without relying on haptic feedback is described in the following.

In the work of Hua and Rifaï [133], the desired velocity of the UAV is adapted so that it does not collide with a single obstacle. This method is extended to several obstacles by Omari et al. [36]. The method requires distance to the obstacles, thus LIDAR sensors are suitable to implement it, as well as cameras. This work relies on a specific control law being applied to the UAV and assumes a velocity controller.

A similar approach is adopted by Hou and Mahony [134], where the distance to an obstacle is used to alter the desired velocity of the UAV so that it stops before hitting an obstacle. A specific controller is not assumed, but the authors show that if perfect velocity tracking is assumed, then obstacle avoidance is guaranteed. It is not clear however what the direction of an obstacle is. An interesting feature of this approach is that, since the

admittance framework is considered, the cues from the environment can be separated from the vehicle's dynamics, giving a clearer representation of the situation to the operator.

In the work of Odelga, Stegagno, and Bülthoff [135], an RGB-D camera is used to create a robot centric, circular grid map of obstacles around the robot. Obstacles are tracked using a bin-occupancy filter. A discrete set of options for the movement of the UAV is defined and model predictive control with a fixed time horizon is used to perform the obstacle avoidance. Similar to the last method, the velocity of the UAV is altered to avoid the obstacles. The runtime of the method is not specified. In particular, the change of runtime with the number of tracked obstacles is absent.

Several conclusions can be drawn from the existing literature. The first one is that obstacle avoidance in teleoperation is often done by changing the input from the user to a safe value. This input is mainly the velocity of the UAV [133–135]. However, the input is generally the position of the surrounding obstacles. Thus, the question that needs to be answered is: given an obstacle position, what velocity should the UAV adopt to not collide with this obstacle? A solution to this problem involves some sort of planning ahead while considering the dynamics of the UAV. Model Predictive Control can be used [135], or other specific control schemes can be adopted [133]. It is argued that this problem can be separated in two different issues: finding a safe position for the UAV and navigating to this safe position. The navigation part of the problem is in fact a position controller, for which a large body of literature exist (e.g., Integral Backstepping [136] or based on geometric methods [24]). It is proposed to decouple those two issues, and focus on the first part of the problem: finding a safe position for the UAV. This way, the obstacle avoidance method is not attached to a specific controller, making the scheme more flexible.

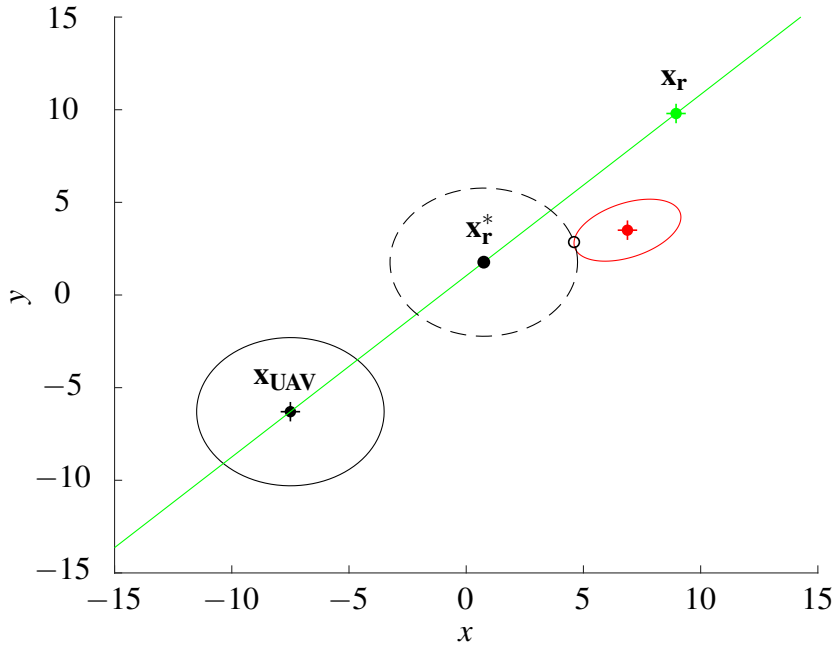


Figure 5.2: Principle of our obstacle avoidance scheme when there is an obstacle on the way

5.3.2 Part 1: obstacle avoidance

5.3.2.1 Problem statement

Given a position of the UAV $\mathbf{x_{UAV}}(t)$, a desired position $\mathbf{x_r}(t)$ generated by the command interpreter (see Section 5.2.2) and a map (generated according to the method described in Chapter 3), the problem is then to find a safe desired position $\mathbf{x_r^s}(t)$ for the UAV. In the following, the UAV is represented as a sphere of radius r_{UAV}.

The general idea is to filter the desired position of the UAV: if there is no obstacle, then $\mathbf{x_r^s}(t) = \mathbf{x_r}(t)$. If there are obstacles on the way, then $\mathbf{x_r^s}(t)$ is placed on the line between $\mathbf{x_r}(t)$ and $\mathbf{x_{UAV}}(t)$, so that the UAV remains in a safe position.

For each obstacle o, a safe position $\mathbf{x_r^*}(t,o)$ is computed, then those positions are fused to create a safe position $\mathbf{x_r^s}(t)$ for all obstacles.

The situation is illustrated in Figure 5.2, with the time and obstacle parameters dropped for clarity. In this figure, the UAV is depicted in black, along with a single obstacle represented by a red ellipse. Here, the obstacle prevents the UAV from reaching

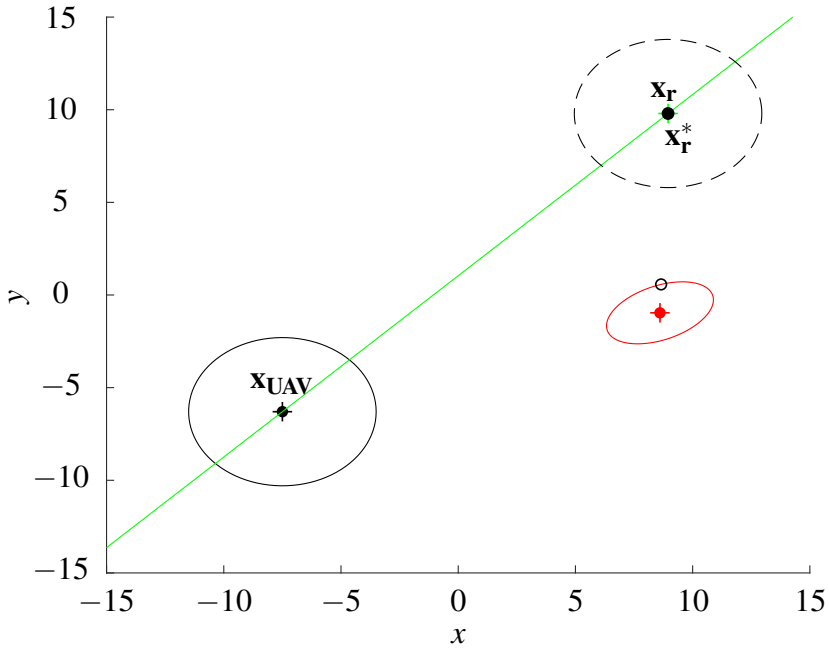


Figure 5.3: Principle of our obstacle avoidance scheme when there is no obstacle on the way

$\mathbf{x_r}$. In this situation, the UAV should stop before it collides with the obstacle. The safe position of the UAV is drawn as a dotted black circle, centred on $\mathbf{x_r^*}$. If the obstacle is far enough from the trajectory of the UAV, then the safe position is centred on $\mathbf{x_r^*}$, as shown in Figure 5.3.

If it is possible to calculate a safe position $\mathbf{x_r^*}$ for each obstacle, then taking the most restrictive position over the set of all obstacles gives an overall collision free position. Let this position be $\mathbf{x_r^s}$. This safe position is then transmitted to the position controller.

5.3.2.2 Representation of the obstacles as ellipsoids

As described in Chapter 3, the map employs normal distributions to represent obstacles, it is then necessary to extract a geometric representation for those obstacles. However, a Gaussian distribution has no boundaries. Thus, the geometric definition of an obstacle can be defined by a cut off distance. Then, each point of the space closer to the mean of the normal distribution than the cut off distance is considered part of the obstacle. In order to determine this cut off distance, a user defined portion $c \in [0,1]$ of the integral of the

probability density function can be used. It is then necessary to compute the cumulative distribution function.

The cumulative distribution function of a multivariate normal distribution has no universal definition. In this chapter, the cumulative distribution function of a multivariate normal distribution is expressed using the Mahalanobis distance r_m:

$$r_m(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{x}) = \sqrt{(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}, \tag{5.18}$$

with $\mathbf{x}$ a sample, $\boldsymbol{\mu}$ the mean of the distribution and $\boldsymbol{\Sigma}$ the covariance of the distribution. If $\mathbf{X}$ is a random vector following a multivariate normal distribution, the cumulative distribution function of this multivariate normal distribution $F(r)$ is defined as:

$$F(r) = \mathbb{P}(r_m(\mathbf{X}) \leq r). \tag{5.19}$$

It is possible to show that $r_m^2(\mathbf{X})$ has a chi-square distribution with k degrees of freedom, where k is the dimension of the vector space (3 in our case). Let us consider a confidence level $c \in [0,1]$. Then, the corresponding Mahalanobis distance r_L so that the probability $\mathbb{P}(r_m(\mathbf{X}) \leq r_L)$ is c is expressed by:

$$r_L = \sqrt{chi2inv(c,3)}, \tag{5.20}$$

where *chi2inv* is the chi-square inverse cumulative distribution function, here with three degrees of freedom.

For a given r_L, an obstacle is thus defined from a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as the set of points $\mathbf{x}$ so that: $r_m(\mathbf{x}) \leq r_L$.

5.3.2.3 Definitions

In the following, $\mathbf{x}_\mathbf{r}^\mathbf{s}$ is formally defined.

Let $\mathcal{G}$ be a list of n Gaussian distributions with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$: $\mathcal{G} = [\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)]_{i \in [\![1,n]\!]}$. Those distributions represent the obstacles and are computed according to the method described in Chapter 3.

The direction $\mathbf{d}$ is defined as:

$$\mathbf{d} = \mathbf{x_r} - \mathbf{x_{UAV}}. \tag{5.21}$$

Note that the direction is not normalised.

5.3.2.4 Computation of the collision free position of the UAV

5.3.2.4.1 Intersection between a sphere along a line and an ellipsoid. Since $\mathbf{x}_\mathbf{r}^*$ is by definition on the line between $\mathbf{x_{UAV}}$ and $\mathbf{x_r}$, the candidates for $\mathbf{x}_\mathbf{r}^*$ are defined as:

$$\mathbf{x}_\mathbf{rp}^*(\lambda) = \mathbf{x_{UAV}} + \lambda \mathbf{d}, \tag{5.22}$$

with λ a real parameter. Note that λ encodes the position on the line between the UAV and the position goal $\mathbf{x_r}$: $\lambda = 0$ corresponds to the position of the UAV and $\lambda = 1$ corresponds to $\mathbf{x_r}$. A range $[\lambda_{\min}, \lambda_{\max}]$ is defined so that if $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ then $\mathbf{x}_\mathbf{rp}^*(\lambda)$ is a safe position of the UAV. Finding the maximal safe position of the UAV is then finding $\lambda_{\max}$, which is the subject of this section.

The strategy used is to loop over all the obstacles and update the values of $\lambda_{\min}$ and $\lambda_{\max}$. More precisely, $\lambda_{\min}$ and $\lambda_{\max}$ are initialised at -1 and 1 respectively. Then, for each obstacle indexed by i, a safe range $[\lambda_{\min}^i, \lambda_{\max}^i]$ is computed and the global range is

updated as:

$$\begin{aligned} \lambda_{\min} &= \max(\lambda_{\min}, \lambda^i_{\min}), \\ \lambda_{\max} &= \min(\lambda_{\max}, \lambda^i_{\max}). \end{aligned} \tag{5.23}$$

Thus, $\lambda^i_{\min}$ and $\lambda^i_{\max}$ have to be computed for a single obstacle. Let us consider a single obstacle with mean $\boldsymbol{\mu}_i$ and covariance $\boldsymbol{\Sigma}_i$. The objective is to find λ so that the point $\mathbf{x}^*_{\mathbf{rp}}(\lambda)$ is at distance r_{UAV} from the ellipsoid extracted from $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. Let $d_e(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, r_l)$ be the distance between a point x and an ellipsoid of mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$ and a threshold r_L. The exact solution λ satisfies the system of equations:

$$\begin{cases} \mathbf{x}^*_{\mathbf{rp}}(\lambda) &= \mathbf{x}_{\mathbf{UAV}} + \lambda \mathbf{d}, \\ r_{\text{UAV}} &= d_e(\mathbf{x}^*_{\mathbf{rp}}(\lambda), \boldsymbol{\mu}, \boldsymbol{\Sigma}, r_l). \end{cases} \tag{5.24}$$

Let us first consider the problem of computing the function d_e. As noted in the work of Hart [137], solving d_e involves solving a polynomial of degree 6 which cannot be done analytically. Iterative methods are however available as a Newton method is proposed by Hart [137] and a bisection method by David Eberly[1]. An approximation can be computed quickly (around 3 iterations) in the 2D case by using the evolute of the ellipse[2]. Once an iterative solution is chosen to compute d_e, equation (5.24) can be solved using an iterative solver. It has to be noted that in general, equation (5.24) has 2 solutions (one before the obstacle, one after).

5.3.2.4.2 An approximated analytical solution. Given the real time constraints, an analytical solution would be beneficial. A popular method to perform obstacle avoidance is to increase the size of the obstacle by adding the radius of the robot. The proposed solution is to approximate the resulting increased obstacle by an ellipsoid. It is then

[1] www.geometrictools.com/Documentation/DistancePointEllipseEllipsoid.pdf
[2] wet-robots.ghost.io/simple-method-for-distance-to-ellipse/

efficient to compute the intersection between a line and this approximation of the obstacle.

Let us consider the set of point $\mathcal{S}_{r_{\text{UAV}}}$ defined by:

$$\mathcal{S}_{r_{\text{UAV}}} = \{\, \mathbf{x} \mid d_e(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, r_l) = r_{\text{UAV}} \,\}. \tag{5.25}$$

In other words, this is the set of all points that are at the exact distance r_{UAV} from the ellipse. This set of points does not form an ellipsoid.

5.3.2.4.3 Ellipsoidal approximation of $\mathcal{S}_{r_{\text{UAV}}}$. To observe this, let us suppose that $\mathcal{S}_{r_{\text{UAV}}}$ forms an ellipsoid. For reasons of symmetry, the ellipsoid formed by $\mathcal{S}_{r_{\text{UAV}}}$ should have the same mean $\boldsymbol{\mu}$ and eigenvectors extracted from $\boldsymbol{\Sigma}$, only the eigenvalues would change. Without loss of generality, let us consider an ellipsoid centred at the origin whose eigenvectors are the canonical basis of $\mathbb{R}^3$:

$$\frac{x^2}{\lambda_1} + \frac{y^2}{\lambda_2} + \frac{z^2}{\lambda_3} = 1, \tag{5.26}$$

where λ_1, λ_2 and λ_3 are the eigenvalues of the covariance matrix associated with the ellipsoid. $\mathcal{S}_{r_{\text{UAV}}}$ is aligned with the canonical basis of $\mathbb{R}^3$ and 3 points are selected on its surface. Since it is assumed that $\mathcal{S}_{r_{\text{UAV}}}$ is an ellipsoid, it is described by equation (5.26) after alignment in the canonical basis. Equation (5.26) is linear in $1/\lambda_i$, the system can thus be solved to get the new eigenvalues corresponding to a new ellipsoid fitted to $\mathcal{S}_{r_{\text{UAV}}}$ using the selected 3 points. If $\mathcal{S}_{r_{\text{UAV}}}$ is truly an ellipsoid, this new ellipsoid should be exactly aligned with $\mathcal{S}_{r_{\text{UAV}}}$. An illustration is provided in 2D in Figure 5.4. The original ellipse, in plain red, has eigenvalues 75 and 2 along the x axis and y axis respectively. The value of r_l has been fixed to $\sqrt{2}$ and r_{UAV} is 2. The increased ellipse, i.e. the set $\mathcal{S}_{r_{\text{UAV}}}$ is plotted in dashed red. In order to describe the fitting points, the parametric mapping of an ellipse is used: $t \mapsto [\sqrt{\lambda_1}\cos(t); \sqrt{\lambda_2}\sin(t)]$ where λ_1 and λ_2 are the eigenvalues

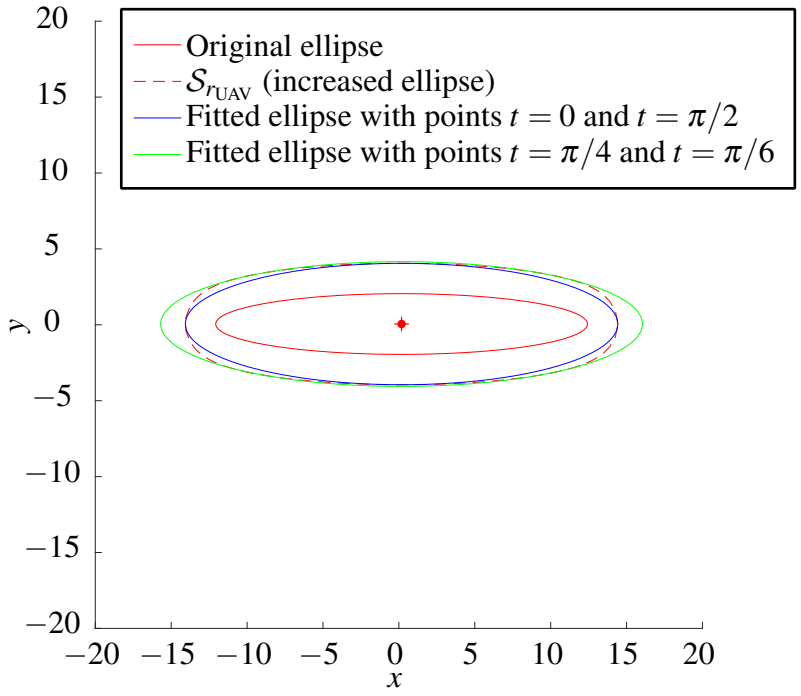


Figure 5.4: Example of increased ellipsoid and its ellipsoid fits for two different set of fitting points

of the covariance matrix. The two points used for the fitting are specified in the legend. Figure 5.4 confirms that $\mathcal{S}_{r_{\text{UAV}}}$ is not an ellipsoid, since it does not coincide with the fitted ellipsoids. The quality of the fit depends on the points chosen. It is shown that a fit using the extreme points of the ellipse (end of each semi great axis, the blue ellipse) tends to underestimate $\mathcal{S}_{r_{\text{UAV}}}$. On the other hand, considering points in between the two semi great axis (the green ellipse) overestimates $\mathcal{S}_{r_{\text{UAV}}}$.

The problem is to find the points (there are two) on a given line that are at a specified distance from an ellipsoid, as described by equation (5.24). Approximating $\mathcal{S}_{r_{\text{UAV}}}$ by an ellipsoid allows us to solve equation (5.24). Given the symmetries exhibited by $\mathcal{S}_{r_{\text{UAV}}}$, an ellipsoid is a reasonable fitting candidate.

Ellipsoid fitting from discrete data points has been studied in the literature [138]. However in this thesis, the problem involves knowledge of both the centre and eigenvectors of the fitted ellipsoid. This makes the method described above attractive due to its computational efficiency.

To summarise: in order to solve the problem described by equation (5.24), a fitted ellipsoid to approximate $\mathcal{S}_{r_{\text{UAV}}}$ is computed. Given a radius r_{UAV}, a probabilistic threshold c, a vector $\mathbf{v}_{\text{ev}}$ containing the eigenvalues of $\mathbf{\Sigma}_i$ and three sets of two angles $\left(\theta_j, \varphi_j\right)_{j\in[\![1,3]\!]}$ (because in 3D, two angles are required to specify an ellipsoid point) the vector $\mathbf{v}^*_{\text{ev}}$ is computed. In the following, this set of three pairs of angles will be referred to as the ellipsoid fitting parameters. It contains the eigenvalues of the covariance matrix describing the increased ellipsoid approximating $\mathcal{S}_{r_{\text{UAV}}}$. The detailed computation is described in Algorithm 5.2.

Algorithm 5.2: Algorithm to compute the eigenvalues of an increased ellipsoid

Input : a vector $\mathbf{v}_{\text{ev}}$ containing the initial eigenvalues,
a confidence level $c \in [0,1]$ for the distribution of the obstacles,
the radius of the UAV r_{UAV},
the ellipsoid fitting parameters (i.e. three pairs of angles) $\left(\theta_j, \varphi_j\right)_{j\in[\![1,3]\!]}$.

Output: a vector $\mathbf{v}^*_{\text{ev}}$ that contains the eigenvalues of the increased ellipsoid.

Function `increaseEllipsoid(`$\mathbf{v}_{\text{ev}}$, c, r_{UAV}, $\left(\theta_j, \varphi_j\right)_{j\in[\![1,3]\!]}$`):`

$r_L^2 = chi2inv(c,3)$;

$\mathbf{D} = \begin{bmatrix} 1/\mathbf{v}_{\text{ev}}(1) & 0 & 0 \\ 0 & 1/\mathbf{v}_{\text{ev}}(2) & 0 \\ 0 & 0 & 1/\mathbf{v}_{\text{ev}}(3) \end{bmatrix}$;

for $i \leftarrow 1$ **to** 3 **do**

$\mathbf{d}_i = \begin{bmatrix} \cos(\theta_i)\cos(\varphi_i) \\ \cos(\theta_i)\sin(\varphi_i) \\ \cos(\theta_i) \end{bmatrix}$;

$\mathbf{p^c}_i = \sqrt{r_L^2 \mathbf{v}_{\text{ev}}(i)} \circ \mathbf{d}_i$;

$\mathbf{p}_i = \mathbf{p^c}_i + r_{\text{UAV}} \frac{\mathbf{Dp^c}_i}{\|\mathbf{Dp^c}_i\|}$;

$\mathbf{q}_i = \mathbf{p}_i \circ \mathbf{p}_i$;

end

$\mathbf{A} = \begin{bmatrix} \mathbf{q}_1^t \\ \mathbf{q}_2^t \\ \mathbf{q}_3^t \end{bmatrix}$;

$\mathbf{k} = \begin{bmatrix} r_L^2 \\ r_L^2 \\ r_L^2 \end{bmatrix}$;

Solve the linear system $\mathbf{Av} = \mathbf{k}$;

$\mathbf{v}^*_{\text{ev}} = \begin{bmatrix} 1/\mathbf{v}(1) \\ 1/\mathbf{v}(2) \\ 1/\mathbf{v}(3) \end{bmatrix}$;

return $\mathbf{v}^*_{\text{ev}}$;

5.3.2.4.4 Finding the collision free position. Let $\mathbf{P}_i$ be the matrix whose columns are the normalized eigenvectors of the covariance matrix $\mathbf{\Sigma}_i$ of the original ellipse. Then the covariance matrix $\mathbf{\Sigma}_i^*$ of the increased ellipsoid is computed as:

$$\mathbf{\Sigma}_i^* = \mathbf{P}_i \operatorname{diag}(\mathbf{v}_{\text{ev}}^*)\mathbf{P}_i^t.$$

The system of equations (5.24) now becomes the intersection between a line and the fitted ellipsoid:

$$\begin{cases} \mathbf{x}_{\mathbf{rp}}^*(\lambda) = & \mathbf{x}_{\mathbf{UAV}} + \lambda \mathbf{d}, \\ r_L^2 = & \left(\mathbf{x}_{\mathbf{rp}}^*(\lambda) - \boldsymbol{\mu}_i\right)^t \mathbf{\Sigma}_i^{*-1} \left(\mathbf{x}_{\mathbf{rp}}^*(\lambda) - \boldsymbol{\mu}_i\right). \end{cases} \tag{5.27}$$

Inserting the first equation into the second one yields a second degree polynomial in λ:

$$\mathbf{d}^t \mathbf{\Sigma}_i^{*-1} \mathbf{d} \lambda^2 + 2\mathbf{d}^t \mathbf{\Sigma}_i^{*-1} \left(\mathbf{x}_{\mathbf{UAV}} - \boldsymbol{\mu}_i\right) \lambda + \left(\mathbf{x}_{\mathbf{UAV}} - \boldsymbol{\mu}_i\right)^t \mathbf{\Sigma}_i^{*-1} \left(\mathbf{x}_{\mathbf{UAV}} - \boldsymbol{\mu}_i\right) - r_L^2 = 0. \tag{5.28}$$

It can be seen that this polynomial has two roots as long as $\mathbf{d}$ is not null. Let us discuss the physical meaning of the possibles roots of this equation.

1. If the equation has no real roots, then there is no collision possible along the direction of travel $\mathbf{d}$: $\lambda_{\min}$ and $\lambda_{\max}$ are not modified.

2. If the equation has two real roots λ_1 and λ_2 of the same sign, then there is a collision on the way. It is assumed without loss of generality that $|\lambda_2| < |\lambda_1|$. If the roots are positive, the collision lies ahead of the UAV, meaning $\lambda_{\max}$ needs to be updated. Then $\lambda_{\max}^i = \lambda_2$. If the roots are negative, $\lambda_{\min}$ needs to be updated. Then $\lambda_{\min}^i = \lambda_2$. In both cases, $\lambda_{\min}$ or $\lambda_{\max}$ is updated according to equation (5.23).

3. If the equation has a single real root, it is treated similarly to the case involving no

roots (a single point of contact has little physical meaning).

4. If the equation has two real roots λ_1 and λ_2 of different signs, it means that the UAV is already inside the increased ellipsoid. While this should not happen with perfect position control, in practice controller overshoot or numerical accuracy issues make this case possible. It is assumed without loss of generality that $|\lambda_2| < |\lambda_1|$. If λ_1 is positive, it means that the UAV is in fact going deeper into the obstacle. In this case, the UAV should go back to safety: $\lambda^i_{\text{max}} = \lambda_2$. Note that this means λ^i_{max} is now negative. If λ_1 is negative, the UAV is leaving the obstacle (but still inside). In this case, $\lambda^i_{\text{min}} = \lambda_2$ is set. Note that this means that λ^i_{min} is now positive. λ_{min} or λ_{max} is finally updated according to equation (5.23).

Let us highlight a small detail about the last case, when the UAV is inside the obstacle. One could argue that it is enough to simply prevent the UAV from going further into the obstacle. It was found that this strategy would sometimes lead to collisions, since it means that the distance between the UAV and an obstacle can never increase, only decrease. The described scheme makes the UAV actively go backward when penetrating obstacles. This behaviour has the following consequence: if the UAV is at the same time between two obstacles and inside those two obstacles, it is possible that $\lambda_{\text{min}} > \lambda_{\text{max}}$. This case means that there is no UAV positions safe range which contains its current position. In practice, such a case would probably lead to a collision, but it is nevertheless necessary to handle it. This particular case is illustrated in Figure 5.5, where the blue increased obstacle constrains λ_{min} to be positive and the red increased obstacle constrains λ_{max} to be negative.

After looping through all the obstacles, the final λ^* to use to get the safe position is computed:

$$\lambda^* = \begin{cases} \lambda_{\text{max}}, & \text{if } \lambda_{\text{min}} \leq \lambda_{\text{max}}, \\ \frac{\lambda_{\text{min}} + \lambda_{\text{max}}}{2}, & \text{if } \lambda_{\text{min}} > \lambda_{\text{max}}. \end{cases} \tag{5.29}$$

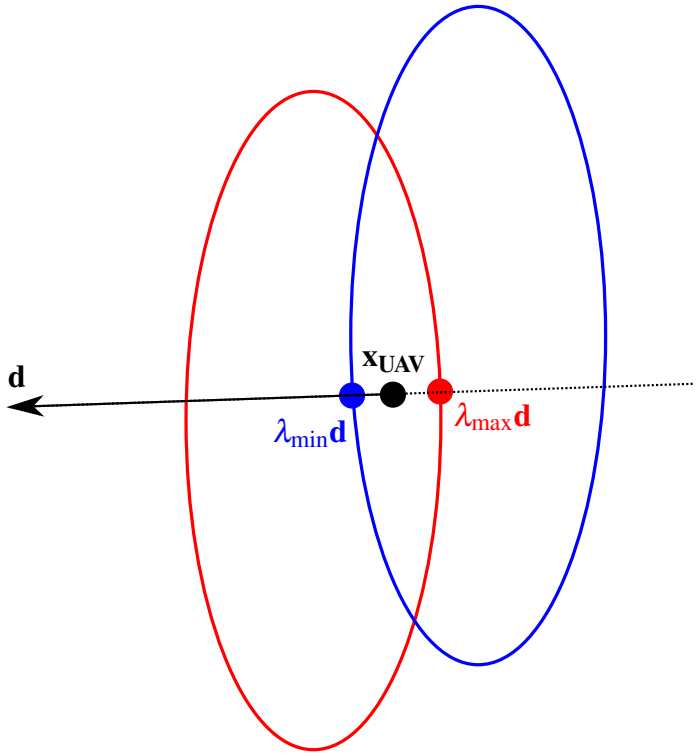


Figure 5.5: Situation where $\lambda_{min} > \lambda_{max}$

The safe position is finally computed:

$$\mathbf{x_r^s} = \mathbf{x_{UAV}} + \lambda^* \mathbf{d}. \tag{5.30}$$

The complete algorithm for computing a safe position is described in Algorithm 5.3. It uses the Algorithm 5.4 to solve the polynomial.

It can be noted that the way the second degree polynomial is solved is meant to avoid loss of precision in the roots. Moreover, t_{sol} cannot be zero if Δ is strictly positive, thus all the divisions are well defined.

5.3.2.5 Speed optimisation: only keep the relevant obstacles

It is unnecessary to use all the obstacles registered in the map. It is possible to select a suitable subset of those obstacles to reduce the computational load. In the proposed implementation, the considered obstacles are either:

Algorithm 5.3: Algorithm to filter the initial position goal $\mathbf{x_r}(t)$ into a safe position goal $\mathbf{x_r^s}(t)$

Input : a position goal $\mathbf{x_r}(t)$,
the position of the UAV $\mathbf{x_{UAV}}(t)$,
a list of normal distributions $\mathcal{G} = [\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)]_{i\in[\![1,n]\!]}$,
a confidence level $c \in [0,1]$ for the distribution of the obstacles,
the radius of the UAV r_{UAV},
ellipsoid fitting parameters (i.e. three pairs of angles) $\left(\theta_j, \varphi_j\right)_{j\in[\![1,3]\!]}$.

Output: a safe position goal $\mathbf{x_r^s}(t)$.

$r_L = \sqrt{chi2inv(c,3)}$;
$\lambda_{\min} = -1$;
$\lambda_{\max} = 1$;
$\mathbf{d} = \mathbf{x_r} - \mathbf{x_{UAV}}$;
if $\|\mathbf{d}\|$ *is too small* **then return** $\mathbf{x_{UAV}}(t)$;
for $i \leftarrow 1$ **to** n **do**
  Compute the eigenvalue decomposition of $\boldsymbol{\Sigma}_i$;
  Store the eigenvectors in $\mathbf{P}$ (one per column);
  Store the eigenvalues in $\mathbf{v}_{\text{ev}}$;
  $\mathbf{v}^*_{\text{ev}} =$ `increaseEllispoid(`$\mathbf{v}_{\text{ev}}$, c, r_{UAV}, $\left(\theta_j, \varphi_j\right)_{j\in[\![1,3]\!]}$`)` ; `// See Algorithm 5.2`
  $\boldsymbol{\Sigma}^*_i = \mathbf{P}\,\text{diag}(\mathbf{v}^*_{\text{ev}})\mathbf{P}^t$;
  $a = \mathbf{d}^t \boldsymbol{\Sigma}^{*-1}_i \mathbf{d}$;
  $b = 2\mathbf{d}^t \boldsymbol{\Sigma}^{*-1}_i (\mathbf{x_{UAV}} - \boldsymbol{\mu}_i)$;
  $c = (\mathbf{x_{UAV}} - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}^{*-1}_i (\mathbf{x_{UAV}} - \boldsymbol{\mu}_i) - r_L^2$;
  $[\lambda_{\min}, \lambda_{\max}] =$ `solvePoly(`a, b, c,$\lambda_{\min}$,$\lambda_{\max}$`)` ; `// See Algorithm 5.4`
end
$\mathbf{x_r^s}(t) = \mathbf{x_r}(t)$;
if $\lambda_{\min} \neq -1 \parallel \lambda_{\max} \neq 1$ **then**
  $\lambda^* = \lambda_{\max}$;
  if $\lambda_{\max} < \lambda_{\min}$ **then**
    $\lambda^* = \frac{\lambda_{\max}+\lambda_{\min}}{2}$;
  end
  $\mathbf{x_r^s}(t) = \mathbf{x_{UAV}} + \lambda^* \mathbf{d}$;
end
return $\mathbf{x_r^s}(t)$;

Algorithm 5.4: Algorithm to solve a second degree polynomial and update the relevant values of λ

Input : coefficients of the polynomial a, b, c, $\lambda_{\min}$ and $\lambda_{\max}$.
Output: $\lambda_{\min}$ and $\lambda_{\max}$

Function `solvePoly(`a, b, c,$\lambda_{\min}$,$\lambda_{\max}$`)`**:**
  $\Delta = b^2 - 4ac$;
  if $\Delta \leq 0$ **then return** $\lambda_{\min}$, $\lambda_{\max}$;
  `// The` *sign* `function returns the sign of its argument.`
  $t_{\text{sol}} = -0.5\left(b + sign(b)\sqrt{\Delta}\right)$;
  $\lambda_1 = t_{\text{sol}}/a$;
  $\lambda_2 = c/t_{\text{sol}}$;
  `// The` *swap* `function swaps its two arguments`
  if $|\lambda_1| < |\lambda_2|$ **then**
    $swap(\lambda_1, \lambda_2)$;
  end
  if $\lambda_1 > 0$ **then**
    $\lambda_{\max} = \min(\lambda_{\max}, \lambda_2)$;
  else
    $\lambda_{\min} = \max(\lambda_{\min}, \lambda_2)$;
  end
  return $\lambda_{\min}$, $\lambda_{\max}$;

1. In a sphere of a certain radius around the UAV r_{fetch}.

2. In a rectangular cuboid starting at the UAV, in the direction of travel, with a length proportional to the current speed using a look ahead time of t^a_{fetch}.

5.3.3 Part 2: navigation assistance

5.3.3.1 Problem statement

At this stage, it is possible to prevent the UAV from colliding with the environment by stopping it before the collision. However, this means that the UAV can be stuck on non planar surface or corners. For instance, if the UAV is blocked against a wall due to the aforementioned scheme, the user has to input a new position at least parallel to this wall, or even make the UAV go backward in order to fly along the wall.

The idea of this section is to design an algorithm for navigation assistance: if there are

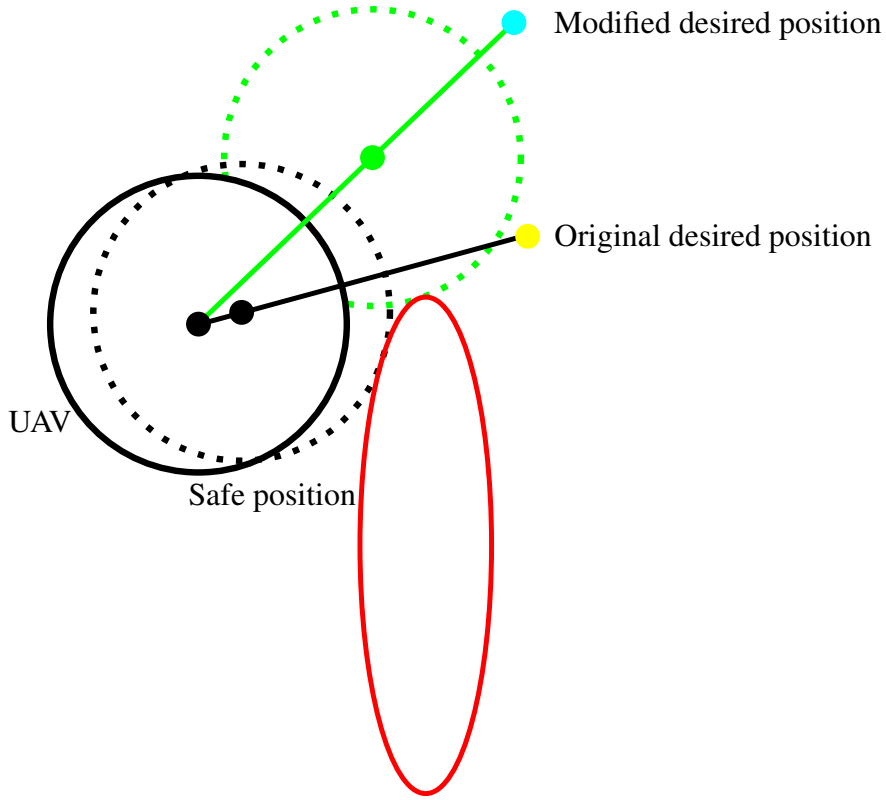


Figure 5.6: Illustration of the general idea of the alternative path finding algorithm

obstacles that prevent the UAV to reach the original desired position, the objective is to compute, should it exists, a new filtered position that better reflects the user input than the safe position $\mathbf{x}_{\mathbf{r}}^{\mathbf{s}}(t)$ computed above. An example is provided in Figure 5.6. The current position of the UAV is in plain black and the desired position in yellow. The obstacle avoidance algorithm described in Section 5.3.2 changes this position in order to avoid collision with the red obstacle. The resulting safe position is plotted in dotted black. In this case, the intent of the user seems to be avoidance of this obstacle on the left side, but the actual input will not achieve this. It is hypothesised that the agility of the UAV would be improved by moving the original desired position to the cyan point. The objective of this section is to describe an algorithm that achieves such modification of the user input.

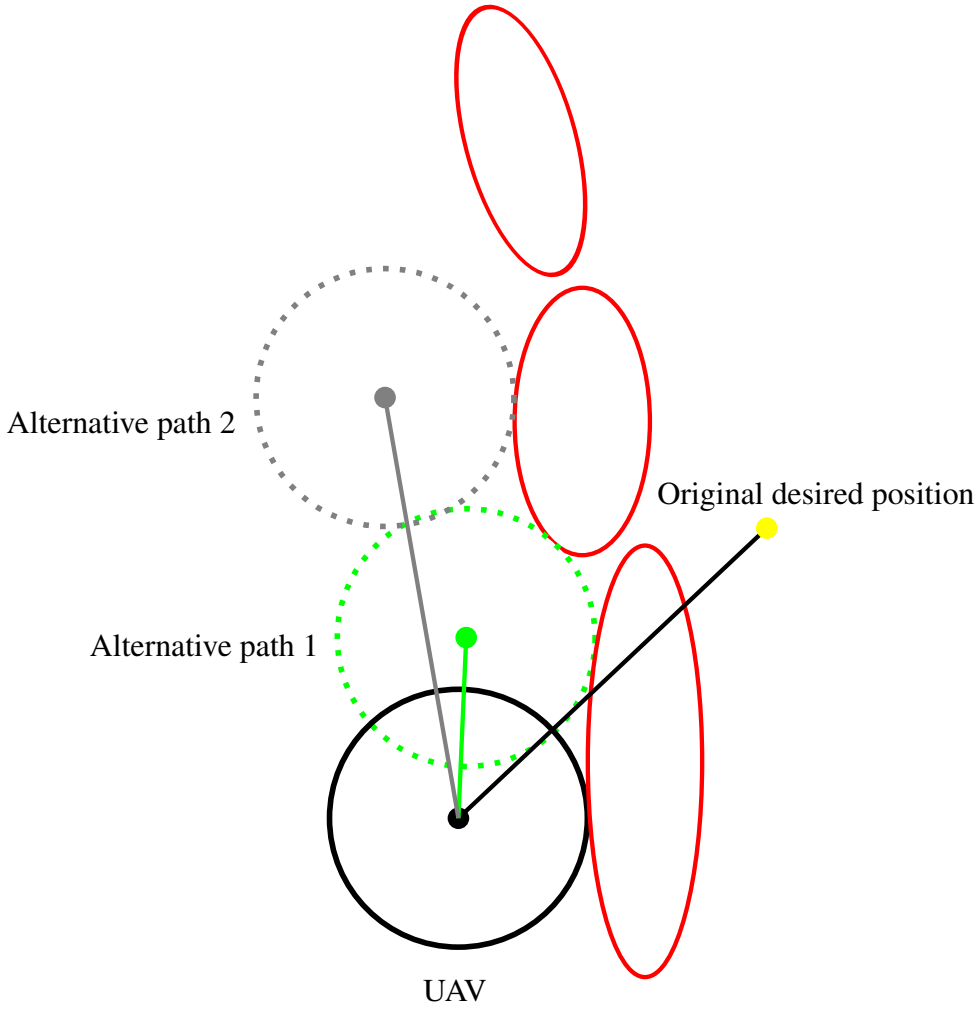


Figure 5.7: A more complex scenario for finding an alternative path

5.3.3.2 Design constraints

The difficulty of designing such an algorithm is that the solution is not always well defined. In the case of Figure 5.6, the solution looks straightforward, but a more complicated example is plotted in Figure 5.7.

The obstacles plotted in red are a typical NDT representation of a wall. The UAV is already at its safe position, for an original desired position represented by the blue dot. The problem is to determine the best alternative path. If the criterion chosen is the proximity with the original goal, a suitable solution would look like the first alternative path in green. If however the criterion is to have a free path of maximum length, the direction would need to be further modified into the second alternative path, in grey.

There is a third option, which is to pass below the obstacles. This solution would provide the shortest path to the original desired position. However, this would mean travelling in a direction that is very different from the one specified originally (the UAV would modify its direction by more than 90°). This is a good illustration of the trade off between optimal path and the user will. Given the teleoperation context, it makes more sense to respect the intent of the user. In practice, this means that the search for an alternative path is limited to a zone reasonably close from the original direction specified by the user. In other words, the user is trusted to give the algorithm a partially faulty input, but no more than a given threshold (a reasonable upper bound would be 90° from the initial direction).

Two main tuning variables can be identified:

- The angle between the original travel direction and the new one.
- The obstacle-free distance along the new direction.

A suitable algorithm would minimise the angle between the two directions while setting the distance along the new direction to a desirable value.

It is important to note that only a change of norm and direction is considered for the direction of travel. In other words, only paths along a straight line are investigated. The main reason for this is the context of teleoperation. Given the high rate of incoming desired positions, more complex trajectories would only be followed for a very short time (until the next user input), which limits their usefulness.

5.3.3.3 Method

In the following, the algorithms are explained in 2D for clarity reasons. The actual implementation is in 3D.

The method is divided in three main steps:

1. Several directions of interest are chosen.
2. For each direction, the distance to the closest obstacle in this direction is computed.

3. Should one of the directions be suitable, it is chosen as the new direction of travel.

In the following, the algorithm is described in more details.

The notations of $\mathbf{x_r}$ for the desired position specified by the operator, $\mathbf{x_{UAV}}$ for the position of the UAV and the direction of travel $\mathbf{d} = \mathbf{x_r} - \mathbf{x_{UAV}}$ are kept in this section.

A maximum angle $\theta_{\max}$ is chosen so that $\theta_{\max} < 90°$. $\theta_{\max}$ represents the maximum angle between $\mathbf{d}$ and the new direction that this algorithm aims to compute. The extreme directions defined by $\mathbf{d}$ rotated by plus or minus $\theta_{\max}$ are called $\mathbf{d}_u$ and $\mathbf{d}_l$ respectively. An angular step θ_s is chosen to discretise the directions between $\mathbf{d}_u$ and $\mathbf{d}_l$. The resulting set of normalized directions is called $\mathcal{S}_\mathbf{d}$.

For each element $\mathbf{d}_i$ of $\mathcal{S}_\mathbf{d}$, the maximum distance λ_i that the UAV can cross before being in collision with an obstacle is computed. Such a computation is performed by using the method described in Section 5.3.2. More specifically, the Algorithm 5.3 is used, but $\mathbf{d}$ is replaced by $\mathbf{d}_i$ and the output is $\lambda_{\max}$ that is initialised at $+\infty$. The output can be thought of as a spherical depth map: for a set of angles, the distances to the closest obstacles are computed. This step is illustrated in Figure 5.8. In this drawing, the limit angle $\theta_{\max}$ is 30° and the step θ_s is 15°. The minimum and maximum directions $\mathbf{d}_l$ and $\mathbf{d}_u$ are $\mathbf{d}_1$ and $\mathbf{d}_5$ respectively. They are defined by a normalized $\mathbf{d}$ rotated by the angle $\pm\theta_{\max}$. The space in between is discretised with step θ_s to create $\mathbf{d}_2$, $\mathbf{d}_3$, $\mathbf{d}_4$ ($\mathbf{d}_3$ is aligned with $\mathbf{d}$ here).

The increased ellipses computed in Section 5.3.2 are represented in red. The distance $\lambda_{i,i\in[\![1,5]\!]}$ to the closest ellipse is computed for each direction $\mathbf{d}_{i,i\in[\![1,5]\!]}$.

Once the directions $\mathbf{d}_i$ and the associated lengths λ_i are computed, a suitable direction $\mathbf{d}^*$ is chosen among the $\mathbf{d}_i$. In order to make this choice, each of the two tuning variables that were identified earlier is examined.

The angle between $\mathbf{d}$ and the chosen $\mathbf{d}_i$ has to be minimal. Moreover, an upper bound λ_i^u can be introduced on the distance along a direction $\mathbf{d}_i$:

$$\lambda_i^u = \mathbf{d}_i \cdot \mathbf{d}. \tag{5.31}$$

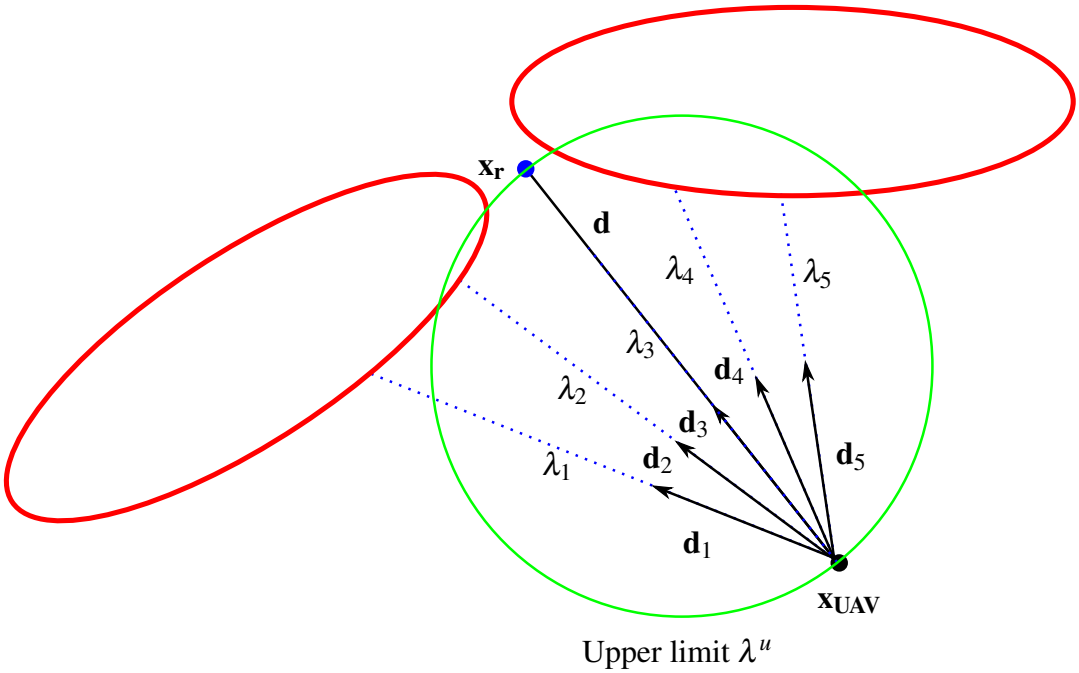


Figure 5.8: Illustration of the alternative directions considered for $\theta_{\max} = 30°$ and $\theta_s = 15°$

Should the length λ_i in a direction be large, this upper limit prevents the UAV from going too far in a given direction, depending on how different this direction is from $\mathbf{d}$. This upper limit has been drawn in green in Figure 5.8 for all possible directions. It is shown that λ_1 is higher than the upper limit, meaning the UAV will be limited in direction $\mathbf{d}_1$, even though the obstacle would allow the UAV to go further.

For all directions $\mathbf{d}_i$, the ratio r_i is computed so that:

$$r_i = \min\left(1, \frac{\lambda_i}{\lambda_i^u}\right). \tag{5.32}$$

If this ratio is equal to 1, it means that the space available in the considered direction has reached the limit. In contrast, if this ratio is less than 1, it means that an obstacle is putting a constraint on the movement in this direction. Since the objective is to search for free paths, the considered directions are all $\mathbf{d}_i$ whose ratio r_i is equal to 1. If there is none, the direction with the highest ratio r_i is chosen. If there are several directions $\mathbf{d}_i$ so that r_i

is equal to 1, then $\mathbf{d}^*$ is the one that is the closest to $\mathbf{d}$ (i.e. the one that maximizes λ_i^u).

In other words, $\mathbf{d}^*$ is the direction that:

- leads to enough space for its λ_i to reach the projection of $\mathbf{d}$ on itself,
- has the smallest angle with $\mathbf{d}$.

Given an initial set of directions and norms $(\mathbf{d}_i, \lambda_i)_{i\in[\![1,q]\!]}$, the computation of the alternative direction $\mathbf{d}^*$ and the length λ^* is detailed in Algorithm 5.5.

Algorithm 5.5: Algorithm to compute an alternative path given a set of obstacles and an original direction

Input : a position goal $\mathbf{x_r}(t)$,
the position of the UAV $\mathbf{x_{UAV}}(t)$,
the radius of the UAV r_{UAV},
a list of directions and the distance to an obstacle in this direction $(\mathbf{d}_i, \lambda_i)_{i\in[\![1,q]\!]}$.
Output: an alternative direction $\lambda^*\mathbf{d}^*$.

```
1  p* = 0 ;                                          // The best projection
2  r* = 0 ;                                          // The highest ratio
3  d = x_r − x_UAV;
4  for i ← 1 to q do
5      λ_i^u = d_i · d;
6      if λ_i^u is too small then
7          continue
8      end
9      r_i = min(1, λ_i / λ_i^u);
10     if r_i ≥ r* then
11         if r* ≥ 1 then
12             if λ_i^u > p* then
13                 p* = λ_i^u;
14                 λ* = λ_i^u;
15                 d* = d_i;
16             end
17         else
18             r* = r_i;
19             λ* = λ_i;
20             d* = d_i;
21         end
22     end
23 end
24 return λ*d*;
```

Once λ^* and $\mathbf{d}^*$ are computed, the new filtered position goal $\mathbf{x}_\mathbf{r}^\mathbf{f}(t)$ is computed as:

$$\mathbf{x}_\mathbf{r}^\mathbf{f}(t) = \mathbf{x}_\mathbf{UAV}(t) + \lambda^* \mathbf{d}^* \tag{5.33}$$

A small note on the extension to 3D: the algorithm has been described in 2D, but the extension to 3D is straightforward. A basis is computed, centred on the UAV, so that $\mathbf{d}$ is the direction of the x basis vector. Then, $\theta_{\max}$ defines the opening angle in the y direction, and $\phi_{\max}$ defines the opening angle for the z direction. The rest of the algorithm is similar.

5.3.3.4 Handling of very close obstacles

It is important to note that this alternative path search only works when the UAV is outside of the increased ellipsoids (i.e. not in potential collision with an obstacle). If this is not the case, then the alternative path search will find no solution. During testing, it was discovered that simply disabling the alternative path search when inside an extended obstacle (and thus relying only on the safe position) produced jerky movements, since the UAV would stop to get back to the safe position before attempting to find a safe path. The proposed solution was to switch the origin of the search. In this section, the origin of the search is the position of the UAV. When the UAV enters inside an increased ellipsoid, the origin of the search is switched to the safe position computed in Section 5.3.2, which is guaranteed to always be outside the increased ellipsoids. This way, a smoother operation of the UAV is achieved.

5.4 Test of the obstacle avoidance module

In order to demonstrate the results of the algorithm described in Section 5.3, several tests are performed in simulated environments.

5.4.1 Adopted parameters, software and hardware

The OAST architecture is implemented as described in Section 5.1.2. The obstacle are detected by a simulated LIDAR scanner, the VLP-16 [94] to which Gaussian noise with zero mean and a standard deviation of 1 cm is added.

Unless mentioned otherwise, the values of all the parameters used in the simulations of this chapter are described in Table 5.1, along with a reference to the equation or section describing them. The algorithms are implemented in C++ as described in Figure 5.1. The interface between sensors are managed using the Robot Operating System (Kinetic) [38] middleware. The operating system is Ubuntu 16.04 64 bits. The simulator used is Gazebo [39], and the package which implements the UAV model is RotorS [40], using a bespoke implementation of the controller [24]. The computer used has an Intel Core i7 6700 processor, an NVIDIA Quadro K2200 graphic card and 16 GB of RAM.

The UAV itself is a Pelican from Asctec, with default characteristics as defined by the RotorS package [40]. The radius of the UAV model is approximately 32.4 cm.

5.4.2 Analysis of the safe position

This first experiment illustrates the safe position computation described in Section 5.3.2. The safe position is investigated in function of two parameters: the noise of the LIDAR sensor and the choice of ellipsoid fitting parameters.

The environment created in Gazebo is only composed of a single wall, as seen in Figure 5.9a. In Figure 5.9b, the normal distributions that form the map are printed in red. If those distributions are potential obstacles as described in paragraph 5.3.2.5 (e.g., they are around the UAV or in the direction of travel), they are coloured in green or orange. In Figure 5.9b, the axes displayed in red, green and blue are the positions of the UAV (below) and LIDAR (above) respectively. The blue coloured ball represents $\mathbf{x_r}(t)$, the position goal of the UAV as sent by the haptic controller. The purple coloured ball represents $\mathbf{x_r^s}(t)$, the safe position of the UAV computed by Algorithm 5.3. In this case,

Table 5.1: Values of the parameters used in the simulation

Parameter	Description	Reference	Value
$\theta_{\max}$, $\phi_{\max}$	Maximum angle for alternative path search	Section 5.3.3.3	80°
θ_s	Angle step between the directions for alternative path search	Section 5.3.3.3	5°
$\mathbf{K}_s$	Scaling matrix converting joystick coordinates into position goal	Equation (5.17)	$\mathrm{diag}(1.0)$
r^*	Radius of the neutral sphere in joystick workspace	Equation (5.17)	0.2
	Radius of the joystick workspace	Equation (5.17)	1
r_{fetch}	Radius of the sphere around the UAV to fetch obstacles	Section 5.3.2.5	2 m
t^a_{fetch}	Look ahead time to fetch obstacles	Section 5.3.2.5	3 s
r_{UAV}	Radius of the UAV with safety margin		60 cm
f_{mp}	Percentage of the maximal force displayed for haptic feedback	Section 5.5	0.3
l_m	Maximal distance between $\mathbf{x_r}$ and $\mathbf{x^f_r}$ considered in haptic feedback computation	Section 5.5	1 m
f_{sp}	Maximal force displayed by inner haptic spring	Section 5.5	1.4 N
c	Probabilistic threshold for computing ellipsoids	Equation (5.20)	0.75
α	Occupancy map update parameter (empty case)	Equation (3.44)	0.45
β	Occupancy map update parameter (occupied case)	Equation (3.44)	0.70
γ	Occupancy map parameter	Equation (3.44)	0.10
	Map width and depth		15 m
	Map height		10 m
	Map resolution		0.3 m
d_{thrc}	Threshold for map recentring	Section 3.3.1	1.5 m
M_{max}	Maximum number of point in a cell during fusion	Section 3.2.4	1000
ε_{rt}	Threshold for finding the next cell during raytracing	Section 3.4	1 mm
K_{occ}	Upper limit on log odd occupancy for a cell	Equation (3.37)	255

the UAV is facing the wall and the user has set a position goal behind this wall. The proposed algorithm has correctly computed the safe position to avoid a collision, which results in the UAV hovering in front of the wall, without advancing further.

In this experiment, the wall is positioned at a distance of 4 m from the UAV. The desired position $\mathbf{x_r}(t)$ does not come from the joystick, but from a program so that:

$$\mathbf{x_r}(t) = \begin{cases} \begin{bmatrix} 0 & 0 & 2t/4 \end{bmatrix}^t, & \text{if } 0\,\text{s} \leq t < 4\,\text{s}, \\ \begin{bmatrix} 5\,(t-4)\,/\,(9-4) & 0 & 2 \end{bmatrix}^t, & \text{if } 4\,\text{s} \leq t < 9\,\text{s}, \\ \begin{bmatrix} 5 & 0 & 2 \end{bmatrix}^t, & \text{if } 9\,\text{s} \leq t. \end{cases} \tag{5.34}$$

In other words, the UAV starts by going up until it reaches an altitude of 2 m, then it goes toward the wall, with the final desired position behind the wall. It is expected that the UAV stays blocked at the wall, which is 2.5 m high.

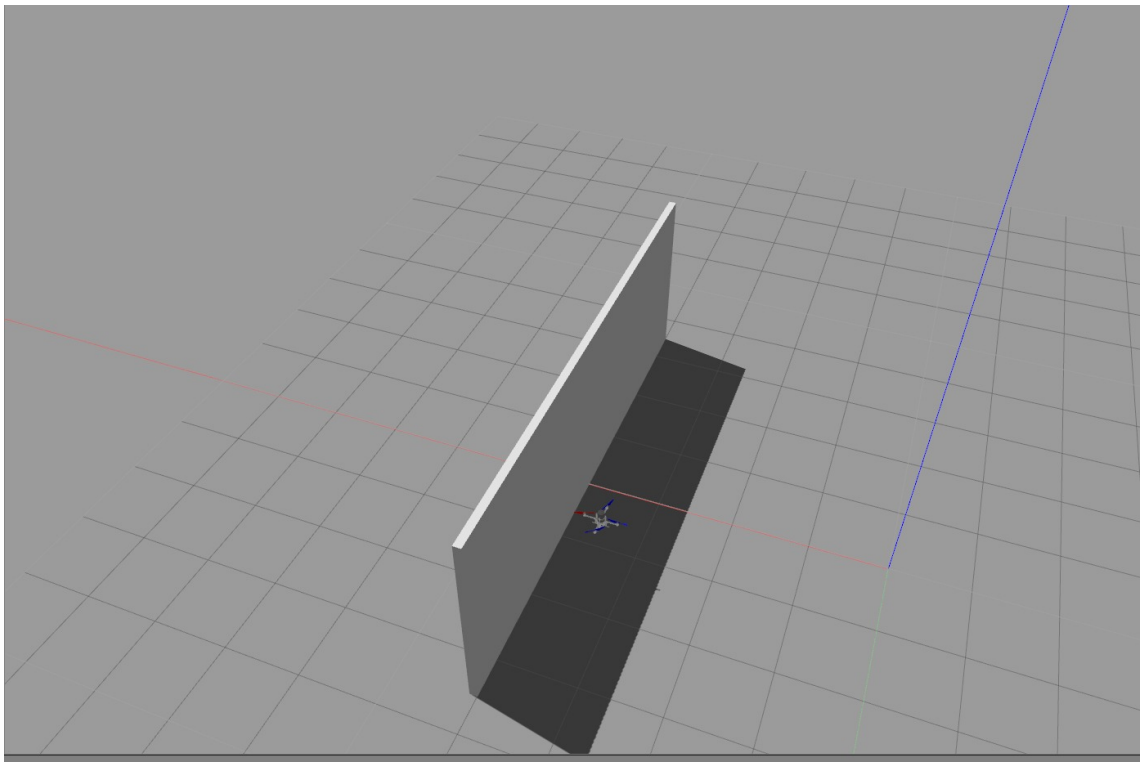
The value of interest is the x coordinate of the safe position $\mathbf{x_r^s}(t)$. In particular, the stability of $\mathbf{x_r^s}(t)$ is examined when the UAV is blocked against the wall for different level of noise in the LIDAR and two choices of points for fitting the increased ellipsoids.

The actual radius of the UAV in our simulation is 38 cm, a security margin is added so that the value of r_{UAV} is 60 cm in this experiment. Thus, the x coordinate of the UAV centre is expected to stay blocked around 3.40 m.

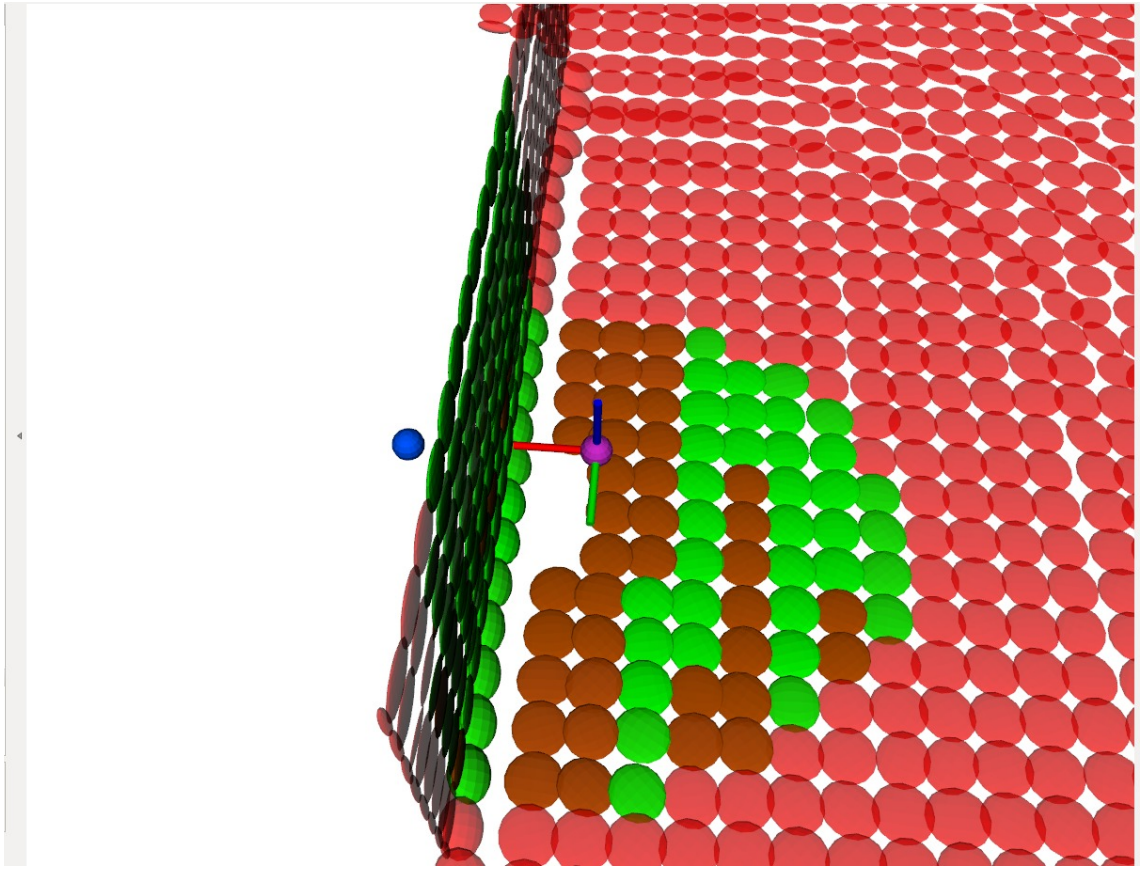
The results of this simulation are shown in Figure 5.10 for two sets of ellipsoid fitting parameters: $(\pi/6, \pi/6)$, $(\pi/6, \pi/3)$, $(\pi/3, \pi/4)$ (called set 1 in the following) and $(0,0)$, $(0, \pi/2)$, $(\pi/2, 0)$ (called set 2 in the following).

It is shown that our algorithm correctly stops the UAV before the obstacle. The precise distance depends on the noise of the sensor: more noise will increase the eigenvalues of the covariance matrices of the obstacles, making the ellipsoids larger and causing the UAV to stop earlier.

It is observed that the stability of the safe position computed is not affected by the



(a) View of the first environment in Gazebo



(b) Gaussian distributions forming the map

Figure 5.9: Pictures of the first simulated environment composed of a single wall

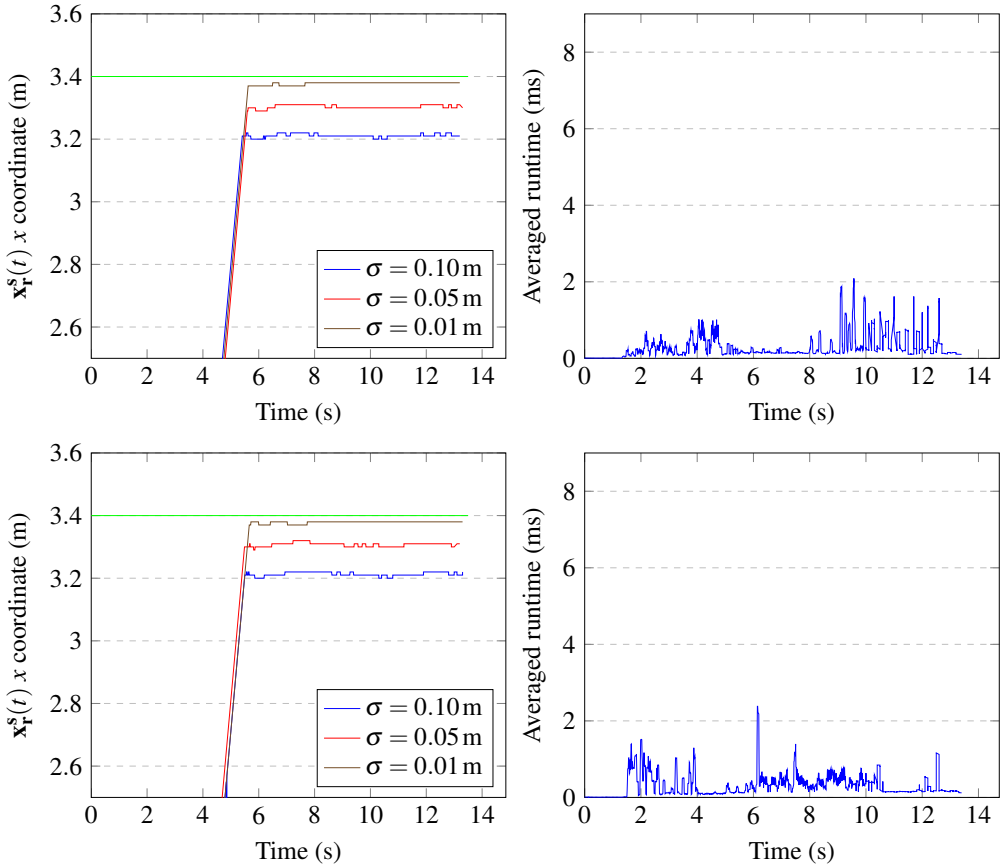


Figure 5.10: Left: Influence of LIDAR noise on the safe position computation. The standard deviation of the noise applied is specified in the legend. Right: Runtime of the safe position computation, a moving average with a window size of 7 is applied. Ellipsoid fitting parameters: $(\pi/6,\pi/6)$, $(\pi/6,\pi/3)$, $(\pi/3,\pi/4)$ (top) and $(0,0)$, $(0,\pi/2)$, $(\pi/2,0)$ (bottom)

increase of LIDAR noise: for the set 1, the standard deviations in the x component of $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$ for $t > 6\,\mathrm{s}$ are 4 mm, 5 mm and 5 mm for LIDAR noise of standard deviation 1 cm, 5 cm and 10 cm respectively. The same behaviour can be seen for the second set of points, with standard deviations in the x component of $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$ for $t > 6\,\mathrm{s}$ of 4 mm, 6 mm and 6 mm for LIDAR noise of standard deviation 1 cm, 5 cm and 10 cm respectively. This result highlights an advantage of the map use compared to the direct use of LIDAR scans: the filtering effect of the NDT creates a stability.

A second advantage of using a map compared to a point cloud is the runtime. Since the map aggregates the points into Gaussian distributions, there are fewer structures to check in the obstacle avoidance method. It is important to highlight that the map update process runs in parallel to the thread that computes the safe position, but both cannot access the map at the same time. Thus, when using the map, the obstacle avoidance process might have to wait for the map update process to end. In this sense, the map update runtime is taken into account in those data. As shown in Figure 5.10, the runtime of the proposed algorithm is usually below 5 ms for this scenario. It is impossible to precisely compare this time to the computation using a raw point cloud since the NDT is mandatory in the approach proposed in this chapter. However, in order to get an approximate idea of the reduction in runtime, an initial version of the current algorithm can be used. This former version was extracting points of interest from an NDT representation, making it comparable to a raw point cloud. For this former scheme, the runtime in this scenario was 8 ms with the map and 40 ms with the raw point cloud.

It is also observed that the choice of fitting points has little influence on the safe position computed in this case: for LIDAR noise of standard deviation 1 cm, 5 cm and 10 cm, the mean x component of $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$ for the set 1 for $t > 6\,\mathrm{s}$ is 3.38 m, 3.30 m and 3.21 m respectively. For the set 2, those values are 3.38 m, 3.31 m and 3.21 m respectively. The result observed, which is a stabilising value of the x component of $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$ around 3.4 m minus two times the LIDAR noise standard deviation, can be explained. The probabilistic threshold for computing the ellipsoids c is set to 0.75, this means that the ellipsoid radius

r_L is around 2.03 as explained in Section 5.3.2.2. Let σ_l be the standard deviation of the LIDAR noise along the x direction. An ellipsoid created from the LIDAR points against the wall is centred at $\mu_x = 4\,\text{m}$. When $y = \mu_y$ and $z = \mu_z$ (i.e. when the furthest point from the mean in the x direction is considered), the equation of the ellipsoid created by the wall is described, along the x direction, by:

$$\frac{(x-\mu_x)^2}{\sigma_l^2} = r_L^2. \tag{5.35}$$

This means that on the x axis, the ellipsoid ends at a distance $\mu_x \pm r_L\sigma_l$. In order to get the stopping coordinate, the radius of the UAV r_{UAV} must be added, so the expected stable value of the x of $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$ is:

$$\mu_x - r_{\text{UAV}} \pm r_L\sigma_l = 3.4 \pm 2.03\sigma_l,$$

which is the observed behaviour.

5.4.3 Analysis of the flight assistance

It is shown in Figure 5.10 that the UAV stops before the wall. In this section, the results of the alternative path finding algorithm proposed in Section 5.3.3 are illustrated. In other words, the filtered position $\mathbf{x}_\mathbf{r}^\mathbf{f}(t)$ is used instead of the safe position $\mathbf{x}_\mathbf{r}^\mathbf{s}(t)$.

5.4.3.1 Alternative path when facing a single wall

The first environment is shown in Figure 5.9a. The trajectory is modified with a vertical component in order to showcase the path finding capabilities. The desired trajectory $\mathbf{x}_\mathbf{r}(t)$

is defined by:

$$\mathbf{x_r}(t) = \begin{cases} \begin{bmatrix} 0 & 0 & 2\min(1, t/8) \end{bmatrix}^t, & \text{if } 0\,\text{s} \leq t < 15\,\text{s}, \\ \begin{bmatrix} 5\,(t-15)\,/\,(20-15) & 0 & 2+(t-15)/10 \end{bmatrix}^t, & \text{if } 15\,\text{s} \leq t < 20\,\text{s}, \\ \begin{bmatrix} 5 & 0 & 2.5 \end{bmatrix}^t, & \text{if } 20\,\text{s} \leq t. \end{cases} \tag{5.36}$$

Note that the wall is 2.5 m high and the maximum desired height caps at 2.5 m as well. This means that the UAV should collide with the wall without any assistance (the UAV should arrive at the wall around the 19 s mark, with an altitude of 2.4 m according to equation (5.36)).

The actual trajectory of the UAV is plotted in Figure 5.11 against the desired trajectory $\mathbf{x_r}(t)$ described by equation (5.36) and the filtered trajectory $\mathbf{x_r^f}(t)$ computed by Algorithm 5.5 and equation (5.33).

The runtime is investigated first. The runtime of a single iteration is typically less than 10 ms for this simple scenario as shown in Figure 5.11. With more obstacles (e.g., if the UAV is inside a very small room with walls all around it), a maximum runtime of around 30 ms has been observed. The runtime increases with the number of obstacles that are checked. Recall that obstacles are checked in the direction of travel, but also in a sphere of radius 2 m around the UAV. This means that obstacles created by the ground are expected to be numerous at first, then decrease as the UAV is gaining height. This is observed in Figure 5.11 between 1 s and 8 s where the runtime, initially high, decreases. The runtime increases again from 16 s as the wall becomes considered by the obstacle avoidance algorithm as described in Section 5.3.2.5.

Although the desired trajectory passes through the wall, the UAV successfully avoids the obstacle. The UAV first starts by advancing toward the wall. At this stage, since there are no obstacles on the way, the filtered position coincides with the desired position. The proposed algorithm modifies the desired trajectory when the UAV arrives too close to the

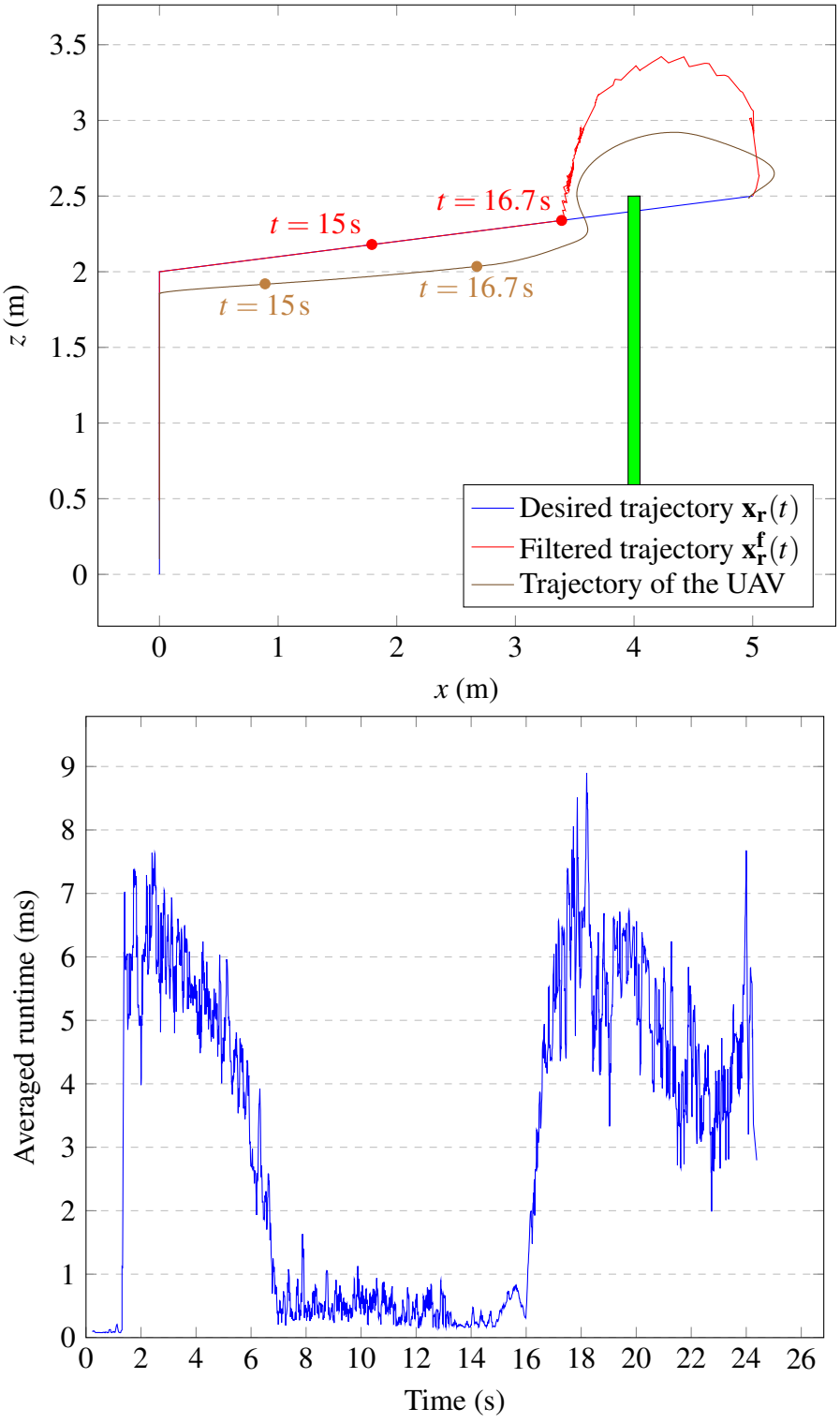


Figure 5.11: Behaviour of the alternative path finding algorithm in a simple scenario involving a single wall. Top: UAV trajectory, input and algorithm output. The wall is plotted in green. Bottom: runtime of the alternative path finding algorithm, a moving average with a window size of 7 samples is applied

wall: since the direction of travel is upward and forward, the closest free path is upward. The OAST architecture thus instructs the UAV to go upward, and tilts the direction of travel forward as soon as the wall does not obstruct this direction anymore.

This simple example highlights a desirable characteristic of the proposed approach: when blocked against a planar surface, the generated trajectories will typically slide against the obstacle in a direction that is close to the desired direction. This makes manoeuvring around obstacles easier and allows the user to be less precise: in a narrow corridor, even if the user does not input a direction of travel perfectly aligned with the walls, the UAV will still advance as if this was the case.

5.4.3.2 Alternative path when following a corridor

Let us illustrate this specific case with a second simulated environment, pictured in Figure 5.12.

In this second environment, there are two parallel walls separated by 1.3 m. The desired position is given by:

$$\mathbf{x_r}(t) = \begin{cases} \begin{bmatrix} 0 & 0 & 2t/10 \end{bmatrix}^t, & \text{if } 0\,\text{s} \leq t < 10\,\text{s}, \\ \begin{bmatrix} 7(t-10)/(16-10) & -1.1 & 2 \end{bmatrix}^t, & \text{if } 10\,\text{s} \leq t < 13\,\text{s}, \\ \begin{bmatrix} 7(t-10)/(16-10) & 1.1 & 2 \end{bmatrix}^t, & \text{if } 13\,\text{s} \leq t < 16\,\text{s}, \\ \begin{bmatrix} 7 & 0 & 2 \end{bmatrix}^t, & \text{if } 16\,\text{s} \leq t. \end{cases} \tag{5.37}$$

In other words, the UAV takes off then goes forward, but with a lateral component that changes direction at $t = 13\,\text{s}$. The trajectory of the UAV is presented in Figure 5.13.

It is shown that in this case, the UAV remains in between the two walls despite the desired position being behind the walls. Moreover, the UAV slides along the walls as discussed above. Those two environments illustrate an increased agility compared to the computation of the safe position only, where the UAV would stop and stay blocked behind

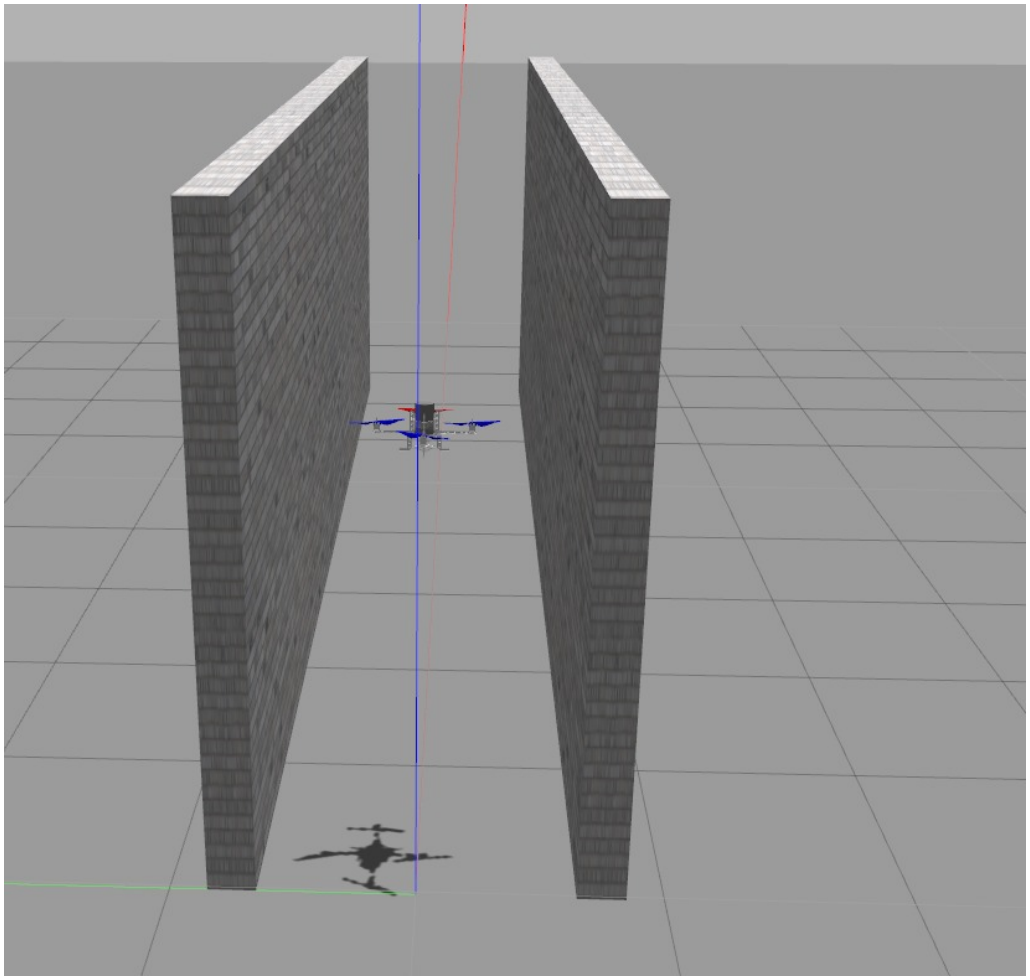


Figure 5.12: Picture of the second simulated environment composed of two parallel walls

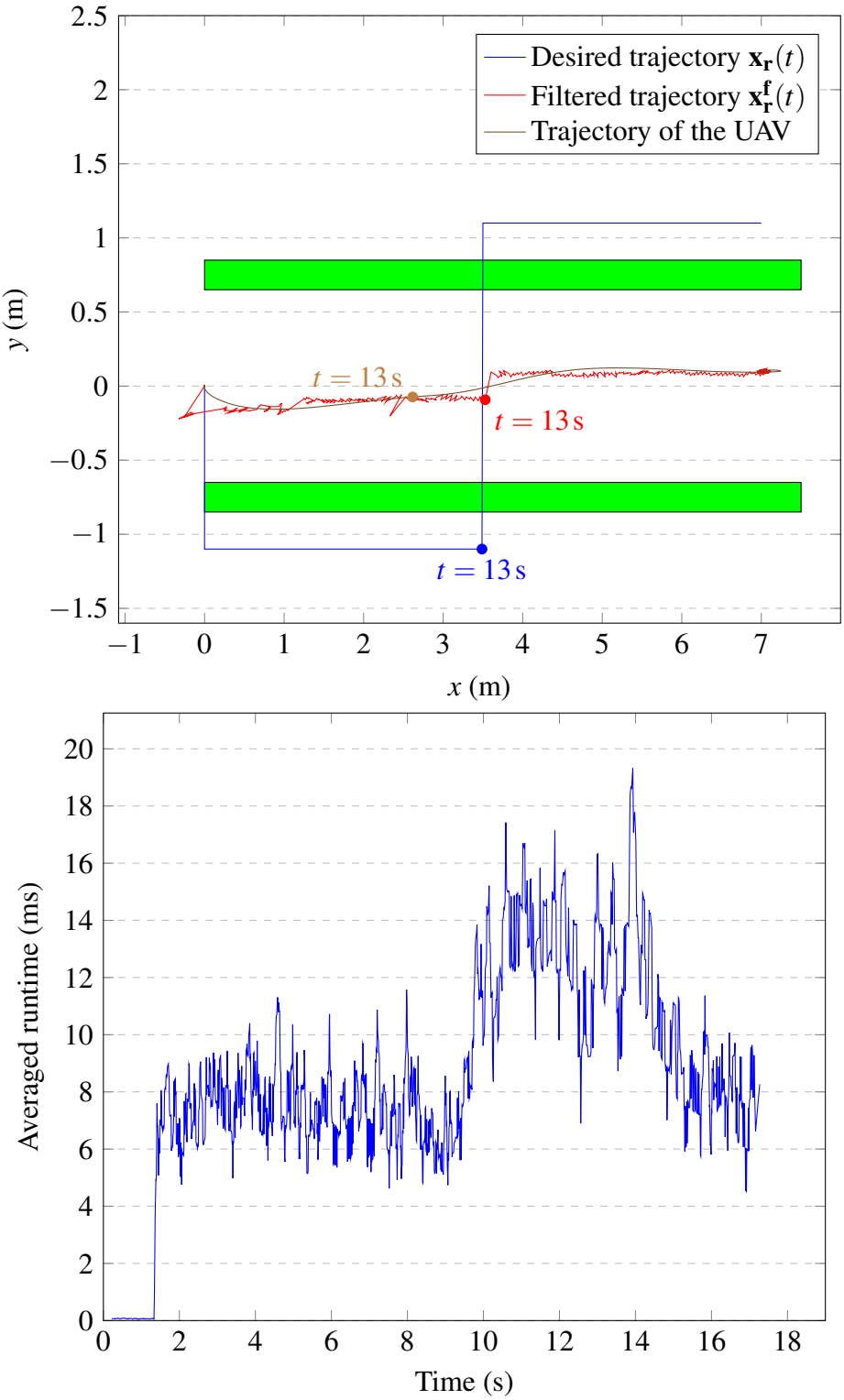


Figure 5.13: Behaviour of the alternative path finding algorithm in a simulated environment involving two walls. Top: UAV trajectory, desired position and filtered position. The walls are plotted in green. Bottom: runtime of the alternative path finding algorithm, a moving average with a window size of 7 samples is applied

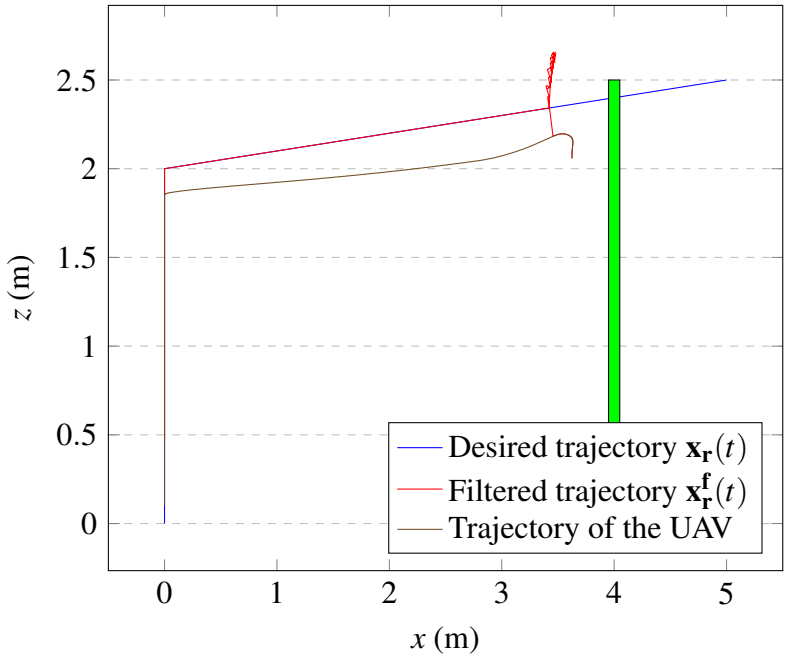


Figure 5.14: UAV trajectory and desired trajectory in a simulated environment without moving the origin of the alternative path finding algorithm. The wall is plotted in green

the obstacle.

5.4.3.3 Alternative path with a fixed origin of the algorithm

A third experiment is proposed to illustrate Section 5.3.3.4 regarding the need to move the starting point of the algorithm seeking an alternative path. It was argued that without moving this starting point to the safe position when the UAV is too close to an obstacle, the displacement would be jerky or even blocked. In order to show this behaviour, the first environment shown in Figure 5.9a is used again, composed of a single wall. The desired trajectory is still described by equation (5.36).

The resulting trajectory is plotted in Figure 5.14. It is shown that without moving the centre of the alternative path finding algorithm, the UAV gets stuck on the wall and cannot progress upward as it should. Indeed, once the UAV is very close to the obstacle, the alternative path finder algorithm cannot find any path since the cone of search is almost entirely obstructed by the obstacle. This behaviour does not occur when the alternative path finder is allowed to change origin, as shown in Figure 5.11 which presents the same

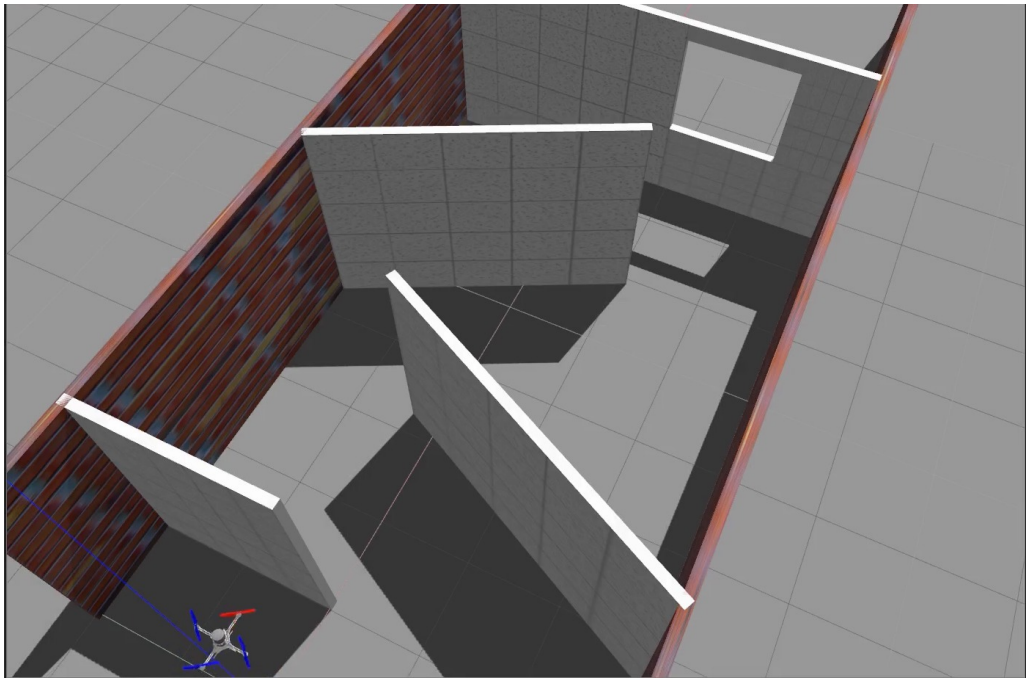


Figure 5.15: Picture of the third simulated environment composed of multiple walls and a window

desired trajectory.

5.4.3.4 Behaviour of the algorithm with a fixed input in a complex environment

Finally, a more complex case is examined. The third environment is depicted in Figure 5.15. This time, a desired position is not provided, but a simulated joystick input is sent to the UAV. This joystick input is simply $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ after take off, meaning that the only command of the UAV is to go forward. Note that the walls are placed in the way of the UAV and the final window available in the wall is slightly shifted compared to the position of the incoming UAV.

The resulting trajectory of the UAV and the filtered position are shown in Figure 5.16. It is shown that the UAV is able to successfully avoid the obstacles on its way. It is possible to differentiate between two types of behaviour from this figure: the very first wall encountered by the UAV is exactly perpendicular to the desired direction of travel. OAST is able to avoid this wall since the free space on the right is inside the cone of search (this zone is represented in Figure 5.8), similar to what is observed in Figure 5.11.

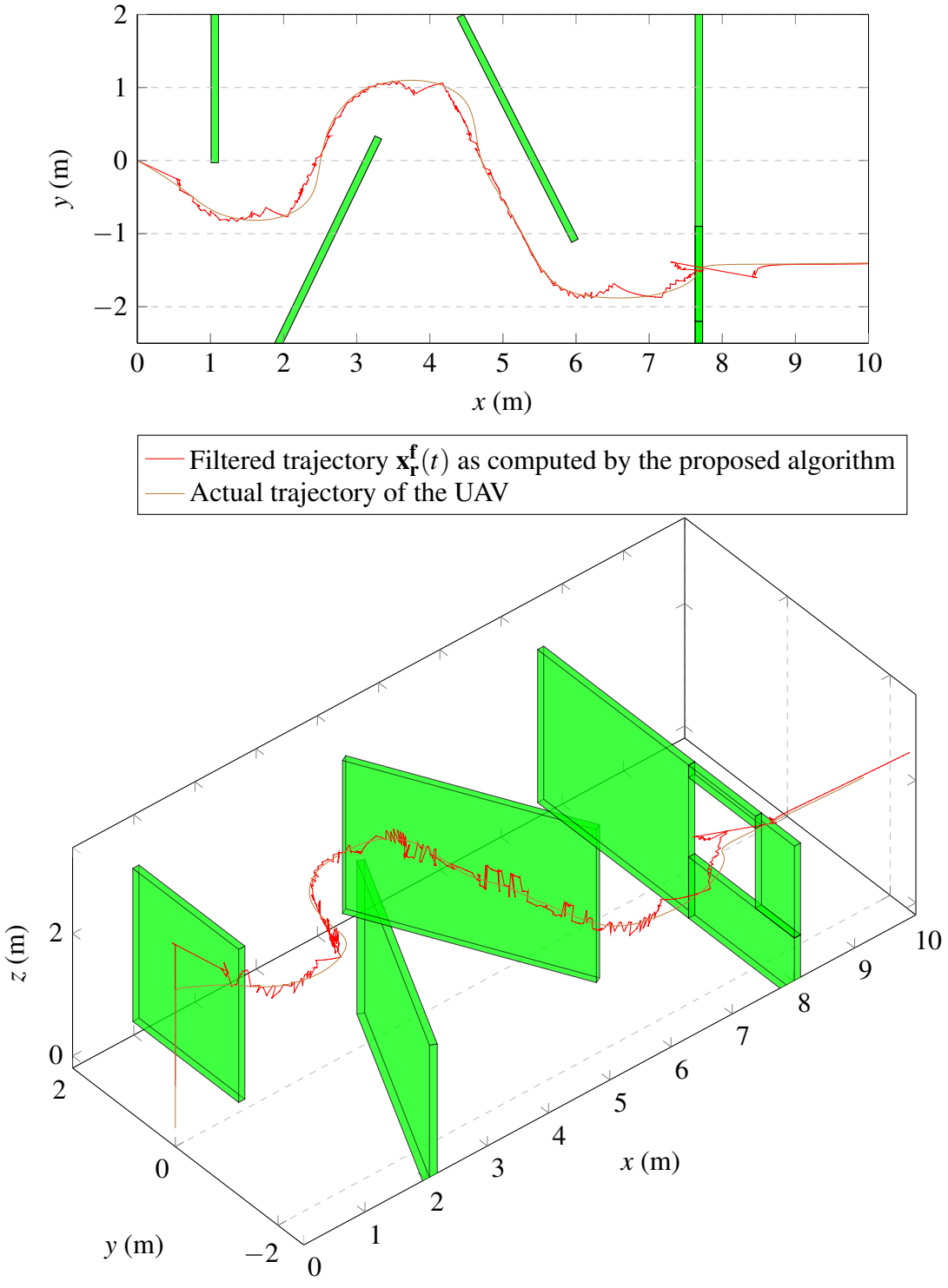


Figure 5.16: Behaviour of the alternative path finding algorithm in the third simulated environment (see Figure 5.15). The trajectory of the UAV is plotted as well as the filtered position computed by our algorithm. The walls are plotted in green except the lateral ones. Two views are shown. The input from the joystick is constant equal to $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$ after take off

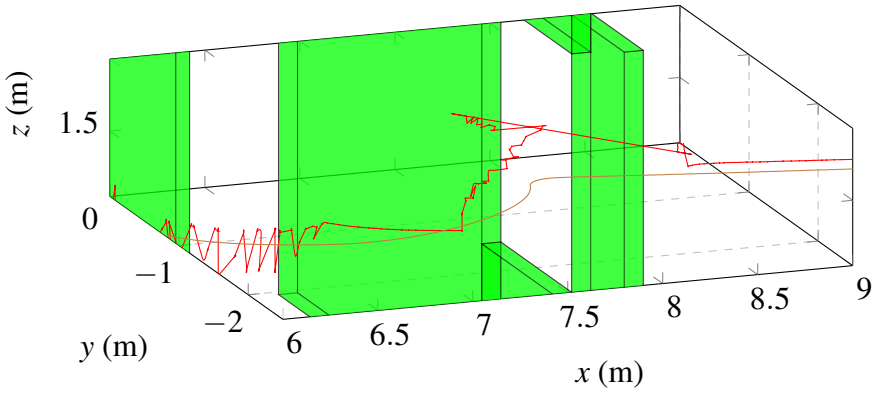


Figure 5.17: Zoomed view of the window passage from Figure 5.16

The second type of behaviour occurs when a solution is not directly visible, which is the case for the second and third walls encountered by the UAV. In those circumstances, the preferred direction computed by Algorithm 5.5 is along the wall since the wall is slightly slanted (similar to what is observed in Figure 5.13). Those results illustrate a limitation of OAST: should the walls be slanted the other way around, the UAV might not pass the obstacle (depending on its position when encountering the walls) and end up in a dead end. This is however by design: since the user might want to actually inspect those dead ends, there is a necessary trade off between the assistance provided by the algorithm and the liberty of movement made available to the user. The results shown in Figure 5.16 illustrate to which degree the user's input is amended. Recall that the input from the joystick is fixed as going forward in the x direction. The 1.3 m wide window is a challenging obstacle with regard to the fixed input provided. A zoomed view around it is shown in Figure 5.17, which shows that our algorithm is able to find the limited zone that allows for the UAV to pass without collision.

5.5 Haptic feedback module

The role of the haptic feedback is different from the one it had in Chapter 2. Before, the feedback was the primary mean of obstacle avoidance. The new objective of the haptic feedback is to facilitate the interaction between the operator and the obstacle avoidance system. Since the proposed obstacle avoidance algorithm modifies the desired position provided by the user, the feedback provides a haptic visualisation of this modification.

More precisely, the feedback is a force proportional to the difference between the desired position $\mathbf{x_r}$ and the filtered position $\mathbf{x_r^f}$. The user can then feel how the algorithm modifies the input, which in turn helps understanding the resulting movement of the UAV.

The haptic feedback uses two tuning parameters: the maximal force that can be displayed f_{mp} (as a percentage of the maximal force that can be displayed by the haptic controller) and the maximal distance l_m between $\mathbf{x_r}$ and $\mathbf{x_r^f}$ that would display this force (in meters). Let $\mathbf{d}_s = \mathbf{x_r^f} - \mathbf{x_r}$. The equation governing the displayed force feedback $\mathbf{f}_u$ is:

$$\mathbf{f}_u = \begin{cases} f_{mp} \frac{\mathbf{d}_s}{l_m}, & \text{if } \|\mathbf{d}_s\| < l_m, \\ f_{mp} \frac{\mathbf{d}_s}{\|\mathbf{d}_s\|}, & \text{else.} \end{cases} \tag{5.38}$$

Moreover, in order to help the operator locating the centre of the workspace, a virtual spring is attached from the centre of the workspace to the end effector. The maximum force displayed by this spring at the end of the workspace is f_{sp}.

5.6 Experiments with human operators

As the proposed flight assistance system is designed for teleoperation, it is important to investigate its impact on human performances. To this end, a human experiment was performed.

5.6.1 Hardware and software

Unless mentioned otherwise, the software and hardware used in this section are described in Section 5.4.1 and the parameters used in our algorithm are described in Table 5.1.

The haptic controller used is the omega.3 from force dimension[3]. Please note that this controller only has 3 degrees of freedom, the yaw is controlled with the keyboard. The library used to interface the haptic controller is HAPI[4] version 1.3 along with the official SDK for the omega.3 version 3.7.3.

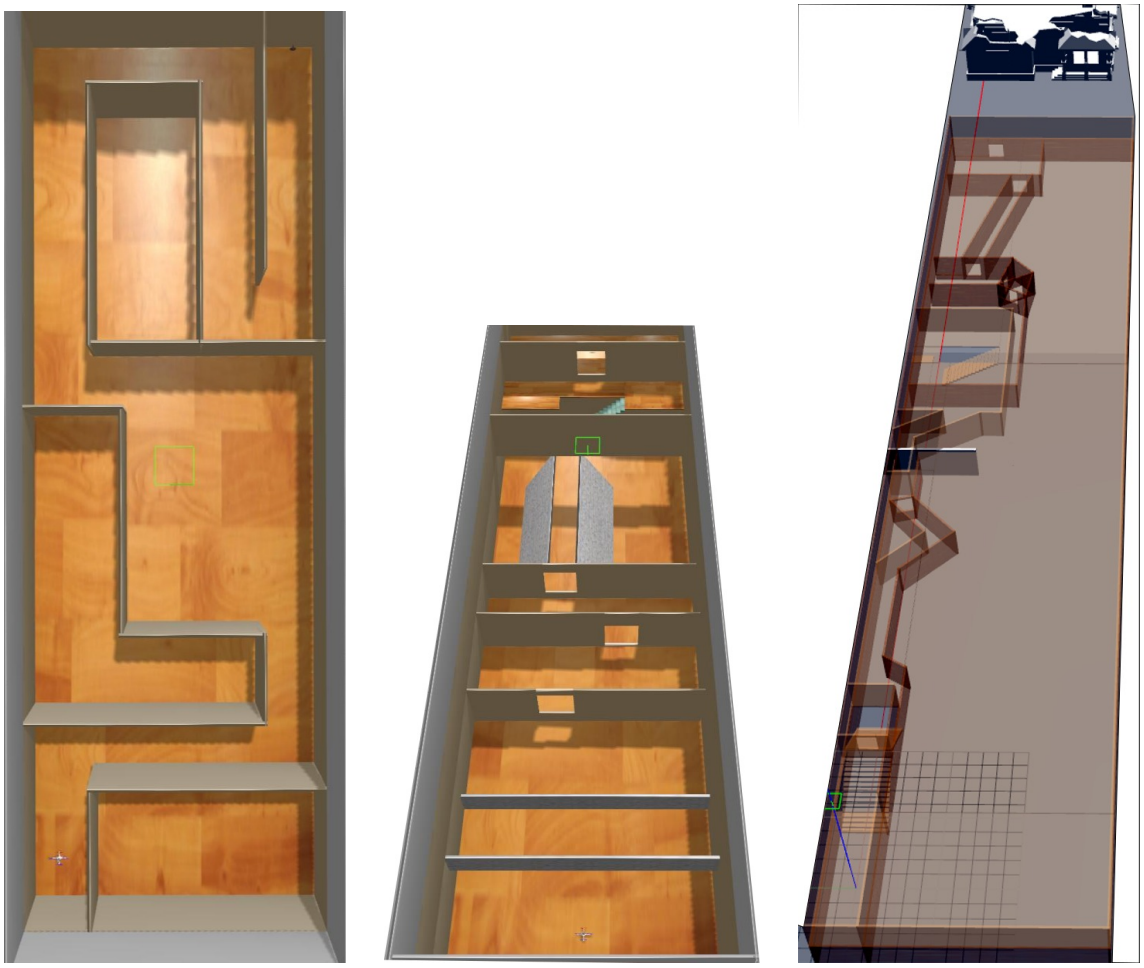
There are 3 sensors mounted on the UAV:

- A forward facing camera with a resolution of 640×480 pixels and a field of view of 80°. This is used by the human operator to operate the UAV.
- A LIDAR with specifications inspired from the VLP-16 [94] of Velodyne. This means it has 16 beams oriented vertically between -15° and 15° (resulting in a vertical angular resolution of 2°). Each beam covers 360° with 1200 points (resulting in a horizontal angular resolution of 0.3°). A Gaussian noise is applied to the range measurement with a 0 mean and a standard deviation of 1 cm. This sensor is used by the obstacle avoidance algorithm as a point cloud source.
- A perfect IMU for ground truth odometry. This sensor is used by the mapping algorithm and the controller.

5.6.2 Scenarios

Three scenarios were developed. The first and second are the same that were used in the experiment described in Section 2.3 while the third one is new. Those scenarios are shown in Figure 5.18. The third scenario is longer and combines elements from the first two. It is expected to be more challenging. Indeed, the windows of the third scenarios are 1.3 m wide while some corridors are only 1.4 m wide. This makes the teleoperation of the UAV

[3] `http://www.forcedimension.com/products/omega-3/specifications`
[4] `https://github.com/SenseGraphics/h3dapi/releases`



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 5.18: Different scenarios used in the experiment (ceiling is not visible on those screenshots)

challenging given its diameter of around 65 cm. Note that in those scenarios, no fusion is performed between the LIDAR and the images from the camera. That is due first to the runtime of the fusion scheme that would not greatly reduce the framerate and second to the visual aspect of the scenarios in Gazebo.

5.6.3 Experimental protocol

The experiments involved 20 people from Cranfield University. Each participant was requested to give informed consent before participating and this research was approved by the Cranfield University Research Ethics System. During the briefing, each participant

was asked, for each scenario, to achieve the following two objectives:

1. minimal number of collisions between the UAV and the environment,

2. be as fast as possible.

Should a collision occur, the UAV would be frozen for 5 s, then brought back where it was 5 s before the collision.

Two schemes are tested: with the proposed collision avoidance system (AS) as described in Section 5.3.3 and without this system (no AS). In other words, in the first case, $\mathbf{x_r^f}$ is sent to the UAV controller while in the second case $\mathbf{x_r}$ is sent instead.

The parameters recorded to assess the performance of the schemes are the number of collisions (N_{col}), the time to complete the run (T_c), the average velocity of the UAV (μ_{vel}), the average distance to the nearest obstacle over a scenario (D_{avg}) and the minimum distance to the nearest obstacle over the whole scenario (D_{min}). The subjective workload is assessed using the NASA Task Load Index (TLX) [41] with pairwise comparison between the weights. A description of the NASA TLX evaluation can be found in annexe A.1.

Each participant is first given 5 min to get familiar with the controls in a training scenario, without the obstacle avoidance system. Then, the obstacle avoidance system (AS) is activated in the training scenario and the user gets 5 min to get familiar with it as well. A test session is composed of 6 runs with or without the obstacle avoidance system: the first three are practice runs (one per scenario) and the last three are the recorded runs. At the end of a test session the participant fills a NASA TLX form for this session. The runs are grouped by session for two reasons: to minimise the learning effect of going through the same scenario back to back, and to allow participants to have a better appreciation of the scheme for the NASA TLX. Each participant does two sessions, one with the obstacle avoidance system and one without. In order to get a full factorial experiment, the number of participants is even and the order of the sessions changes between each participant. This protocol yields 6 runs and 2 NASA TLX evaluations per person, for a total of 120 runs and 40 NASA TLX evaluations. The time allocated per participant was 90 min.

It is important to emphasise a particular property of the number of collisions in this experiment. When the UAV collides with any object in the simulation, it is moved in the position it occupied 5 s before the collision occurred. In the real world, a collision with any part of the propellers is likely to heavily damage or destroy the UAV. A realistic reaction would then be to move the UAV back at the beginning of the scenario, to simulate a replacement of the UAV. The time necessary to perform the experiment would however be too important. Moreover, the tiredness of the participants could become a significant factor in the experiment. It can be argued that the presence of collisions taints the runs where they take place when looking at metrics such as the time of completion. As explained above, the true cost of a collision is difficult to simulate perfectly, but from the point of view of time of completion, it is certainly underestimated in this study. Since the objective of this study is to compare the performance of OAST against what a human would actually do, it does not seem appropriate to attempt to create collisions free trajectories for comparison purposes.

5.7 Experimental results

All the box plots presented in the following are built as follows: the median is reported between the first and third quartile. The top and bottom whiskers maximum length is 1.5 times the interquartile range. They however only extends to the most extreme data point inside this range. Any point outside of the whisker range is plotted individually.

Results were analysed with the R language [42], using generalised linear mixed models with the *lme4* [43] package. Each parameter was modelled separately using the scenarios and methods as fixed factors plus an intercept. The participants were modelled as random effects. The confidence level used for statistical significance is 0.95. The initial model was created by including an interaction term between method and scenario, which was dropped if its contribution to the model was not shown to be significant with the ANOVA. If the algorithm proves having a statistically significant influence, post-hoc tests

Table 5.2: Results of a full factorial ANOVA on the two fixed effects: method and scenario

Fixed effect	N_{col}	D_{min}	D_{avg}	T_c	μ_{vel}	TLX
Method	$***$	$***$	$\cdot$	$***$	$***$	$***$
Scenario	$***$	$**$	$***$	$***$	$**$	*NA*
Interaction method $\times$ scenario	$\cdot$	$\cdot$	$***$	$**$	$\cdot$	*NA*

p-values are indicated by '$***$', '$**$', '$*$' and '$\cdot$' if respectively $p \leq 0.001$, $0.001 < p \leq 0.01$, $0.01 < p \leq 0.05$ and $p > 0.05$ (not significant).

are carried out using package *lsmeans* [44] to determine the nature of this influence per scenario. Unless mentioned otherwise, each generalised linear mixed model is using a family of Gaussian distribution.

The p-values from the full factorial ANOVA are presented in Table 5.2. For this experiment, the method factor has two levels (with the avoidance system and without it) and the scenario factor has three levels.

5.7.1 Safety metrics: number of collisions, average minimal distance to an obstacle and minimal distance to an obstacle

In this section, the results related to the safety of teleoperation are presented through three metrics: the number of collisions, the mean distance from the UAV to the nearest obstacle and the overall minimal distance between the UAV and the obstacles. It has to be noted that the distance to the closest obstacle is computed from LIDAR data, which means that obstacles outside of the LIDAR field of view are not reported with this metric. Moreover and in order to avoid detecting the propeller's blades with the LIDAR, the minimum distance reported by the LIDAR is 35 cm.

The numbers of collisions are presented in Figure 5.19. Those numbers are obtained by accumulating the number of collisions for a given scenario and a given method across all users. This means that each box plot is created from 20 points. The number of collision

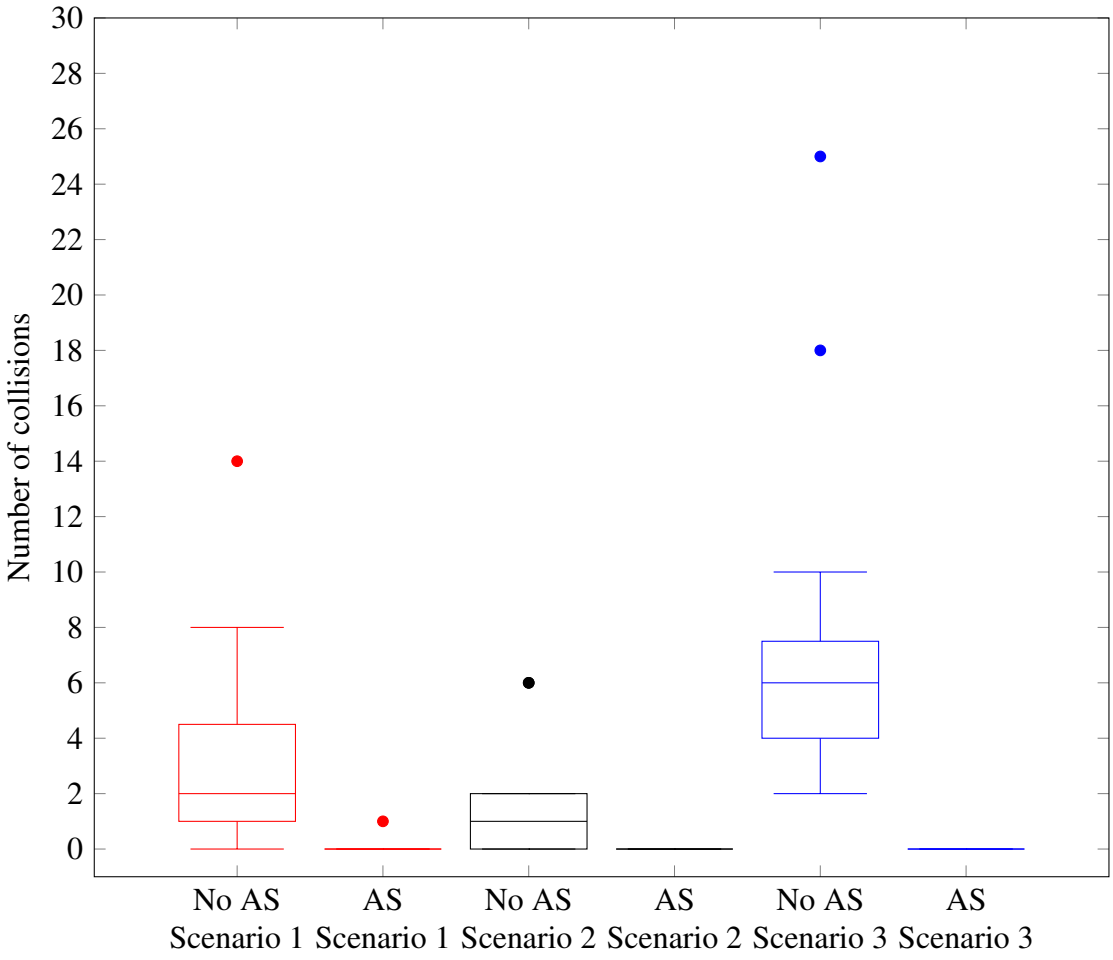


Figure 5.19: Number of collisions with and without the avoidance system. An outlier of 67 collisions in scenario 3 without the avoidance system is omitted from this graph

with the avoidance system activated is always zero, except for one collision in scenario 1. Without the obstacle avoidance system, the number of collisions depends on the scenario. The third scenario, expected to be the most difficult, contains the highest number of collisions. Since the number of collisions is a count, it is modeled as a Poisson distribution. The reduction in collisions provided by the algorithm is statistically significant ($p < 0.0001$) for each scenario. It is interesting to note that while human operators are sensitive to the difficulty of the scenario, the algorithm is not.

Moreover, it is important to remember that the simulated sensor used to scan the environment is a LIDAR which could realistically be embedded on a small UAV [94]. In particular, this LIDAR has a vertical field of view of 30°. This experiment shows that, given perfect odometry, such a restricted field of view still allows obstacle detection in tasks that involve verticality (scenarios 2 and 3 in particular). In other words, the trajectories taken by the participants allowed a sensor with a vertical field of view of 30° to capture enough volume to successfully avoid collisions in a closed environment.

The average distance to the closest obstacle over a completed scenario is presented in Figure 5.20, while the overall minimal distance to the closest obstacle is presented in Figure 5.22. In order to compute the average distance to the closest obstacle, the distance of the closest LIDAR point is averaged per user for a given scenario and method. This gives a single data point. Those data points are then used to create the boxplots in Figure 5.20. On the other hand, to get the overall minimal distance to the closest obstacle, the smallest distance to the closest LIDAR point is kept for a given scenario and method. This gives again a single data point. The data points for each user are then used to create the boxplots in Figure 5.22. As a result, the boxplots in both Figure 5.20 and 5.22 are created from 20 points.

Note that in the case of the overall minimal distance, only the results with the avoidance system are presented, since most of the runs without this system contain a collision, which would correspond to the cutting length of the LIDAR.

The average distance to the closest obstacle shows no clear trend, in contrast with what

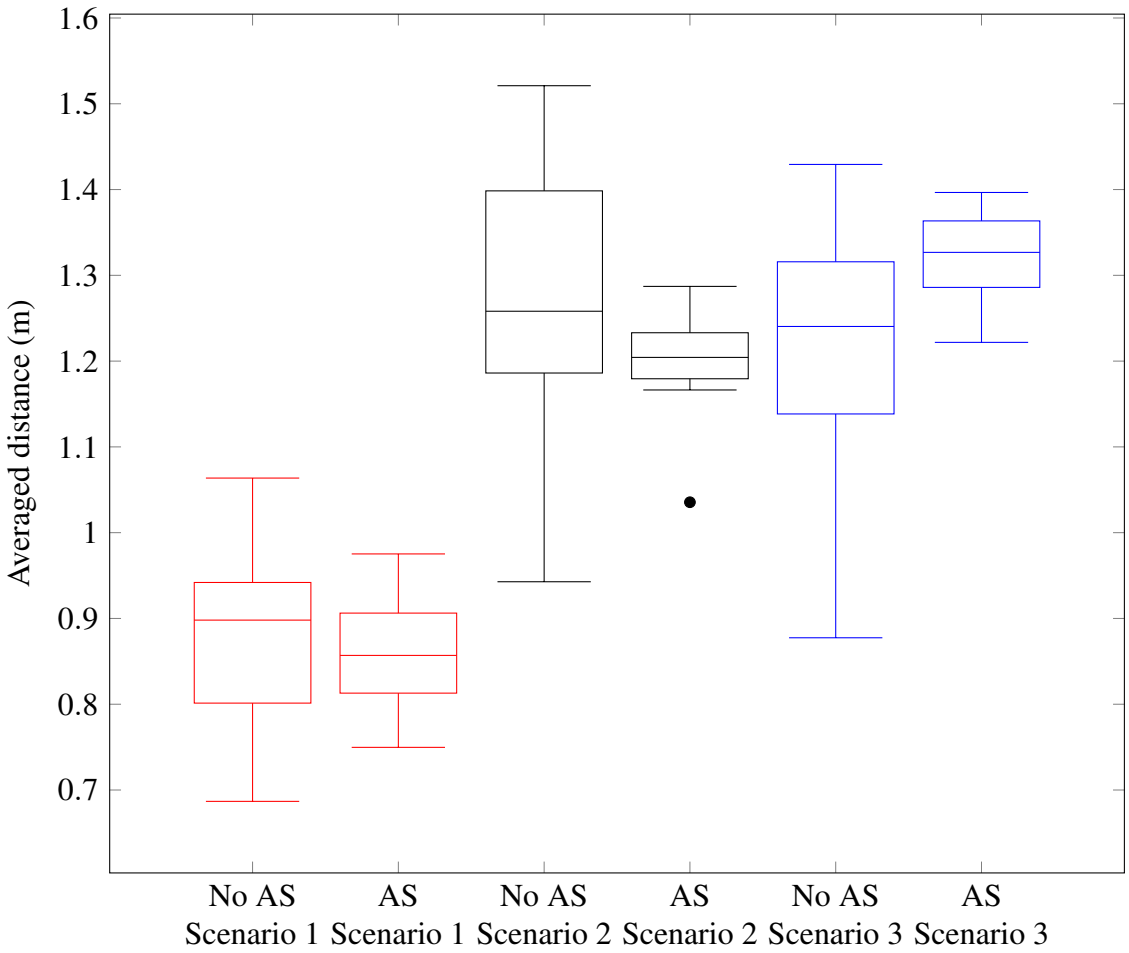
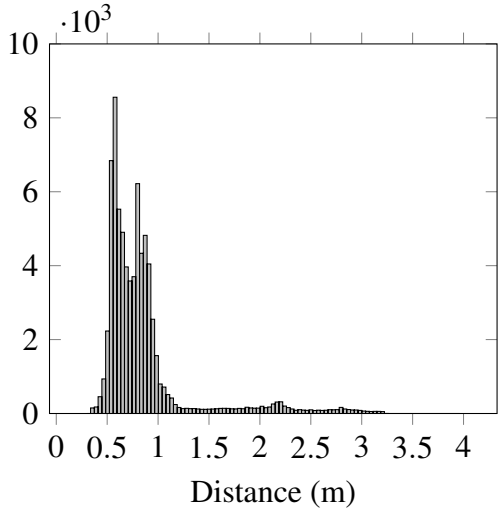


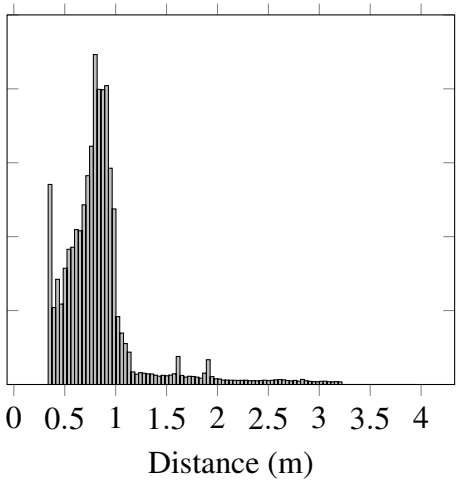
Figure 5.20: Average distance to the closest obstacle with and without the avoidance system as measured by the simulated LIDAR

would be expected if a method based on potential fields was used, which tends to maximise the distance to obstacles. With the obstacle avoidance system, the median is lower in scenario 1 and 2 but higher in scenario 3. The ANOVA shows no significant influence of the algorithm on the average distance to the closest obstacle ($p = 0.82$) although a significant interaction term is present between the presence of the algorithm and the scenario. Post-hoc test reveals a significant difference in scenario 3 by a very small margin (the lower confidence level with the obstacle avoidance algorithm is 1.28 m against an upper confidence level of 1.27 m without the algorithm). The objective is to allow for as much liberty as possible for the operator, which means that the average distance to the closest obstacle should not be expected to increase because the movement pattern of the operator should not change significantly. This distance might decrease if the operator explicitly relies on the avoidance scheme to steer the UAV in the right direction while keeping as close to an obstacle as possible. Such an analysis is difficult to perform from the averaged distance to the closest obstacle alone, so histograms of the distance to the closest obstacle for all participants (again, computed from the LIDAR data) are shown in Figure 5.21. Those histograms show that in all three scenarios, the UAV spends slightly more time close to obstacles when the obstacle avoidance algorithm is activated. This suggests that participants did not only rely on the avoidance system to perform emergency avoidance, they also rely on it to perform trajectory planning.

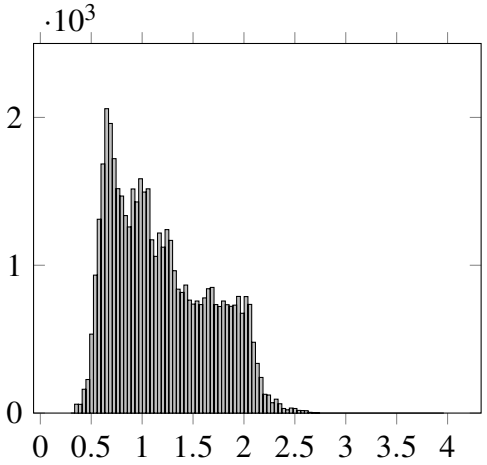
The minimum distance to the closest obstacle gives an idea of the security margin provided by the algorithm. Beside the avoidance system, this security margin is influenced by the behaviour of the user, the topology of the scenario and the overshoot of the UAV controller. As expected due to the almost total absence of collisions with the avoidance system, the influence of the algorithm on the minimum distance to the closest obstacle is significant. As mentioned before, the radius of the UAV is 32.4 cm at its largest point. However, the UAV in the simulation is not a sphere, which means that while a closest obstacle further than 32.4 cm guarantees no collision, a closest obstacle closer than 32.4 cm does not guarantee a collision. Figure 5.22 shows that in 75 % of the runs



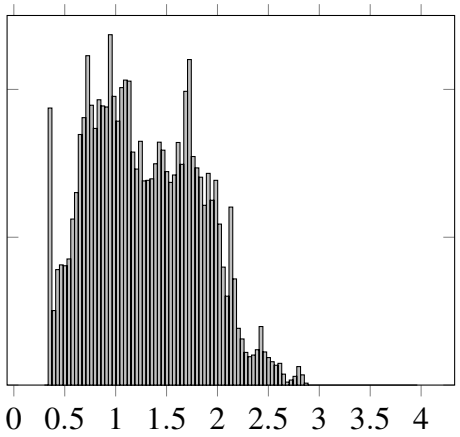
(a) Scenario 1 with AS



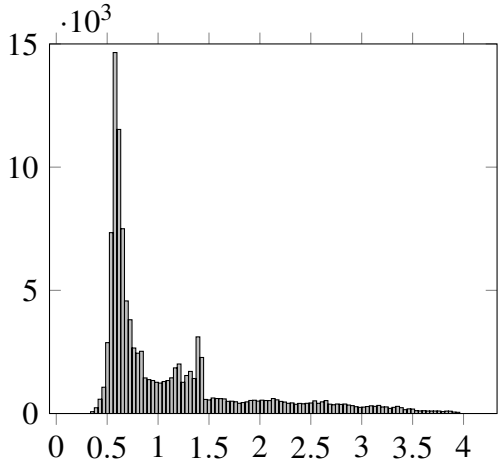
(b) Scenario 1 with no AS



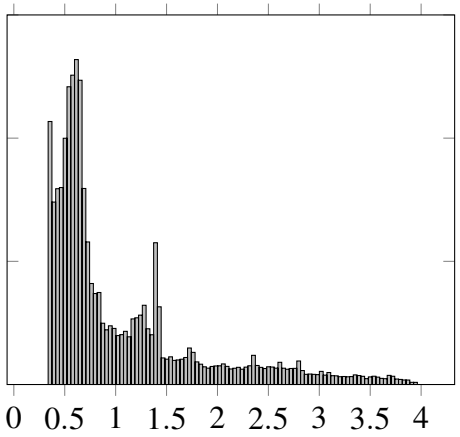
(c) Scenario 2 with AS



(d) Scenario 2 with no AS



(e) Scenario 3 with AS



(f) Scenario 3 with no AS

Figure 5.21: Histograms of the distance to the closest obstacle for all users with (on the left) and without (on the right) the avoidance system as measured by the simulated LIDAR

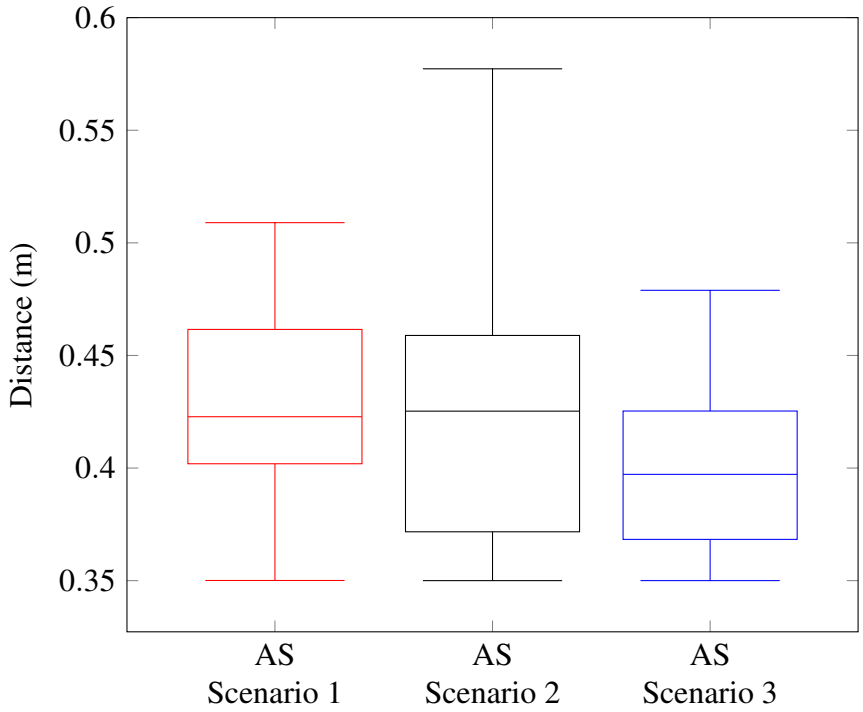


Figure 5.22: Minimum distance to the closest obstacle with the avoidance system over the whole run as measured by the simulated LIDAR

with the obstacle avoidance system, the closest distance to an obstacle detected by the LIDAR was 40 cm, 37 cm and 37 cm for scenarios 1, 2 and 3 respectively. This means a security margin comprised at least between 5 cm and 8 cm is provided by the avoidance system.

In order to have a more qualitative look at the influence of the avoidance system on the adopted trajectories by the participants, a superposition of all the trajectories in the first scenario is presented in Figure 5.23. Note that, in accordance with the experimental protocol described in section 5.6.3, each participant did two recorded runs in this scenario, one with the obstacle avoidance system and one without. Thus, there are two runs per participant shown on this figure. It appears that the participants take tighter turns without the avoidance system, a behaviour that can (and did) lead to collisions. In particular, the very first turn and the last two showcase this trend. This might be explained by the fact that the inner part of the turn remains visible longer during a turn, thus appearing safer than the outer part. The penultimate turn shows that several participants explicitly

relied on the algorithm to stop them before hitting the wall, thus making a 90° turn. A second observation is that the trajectories with or without the avoidance system do not look very different. This is partly due to the relatively linear design of the scenario, but this also highlights the fact that the avoidance system is transparent and do not translate to significant changes in trajectory. This is advantageous since it means that the user keeps a liberty of movement and is not constrained by the algorithm. Instead, the algorithm only operate slight changes when required. Such an example is the fourth turn, where the concentration of red lines show that a lot of participant were saved by the algorithm.

An interesting question when looking at Figure 5.23 is the existence of a reference or ideal trajectory. Such a trajectory can be computed if the objective is to minimise the length of the path taken, however a safe trajectory is a more complicated issue. Indeed, safety cannot be easily quantified: metrics such as minimal distance to an obstacle can be used, but being close to an obstacle is not necessarily unsafe, depending on the velocity of the UAV. The main metric used in this thesis, the number of collisions, cannot be used to create a trajectory since there is an infinity of trajectories that do not collide with the environment. To make the matter worse, safety in teleoperation depends on the skill of the operator as well: some maneuvers can be performed safely by some operators and not others. Moreover, while it is often desirable to go faster, safety has limited benefits above a certain threshold that is application dependent. The instruction given to the participant was to minimise the number of collisions and be as fast as possible. The existence of an ideal trajectory to achieve this objective would require the creation of an order relation between two trajectories from the point of view of safety, which is an open problem. For this reason, an ideal trajectory does not appear on this figure.

5.7.2 Efficiency metrics: time elapsed and average speed

In this section, the metrics related to the efficiency of the teleoperation schemes are presented. The time to complete each scenario is shown in Figure 5.24, each box plot is made from 20 data points. This figure shows that the avoidance system reduces the time needed

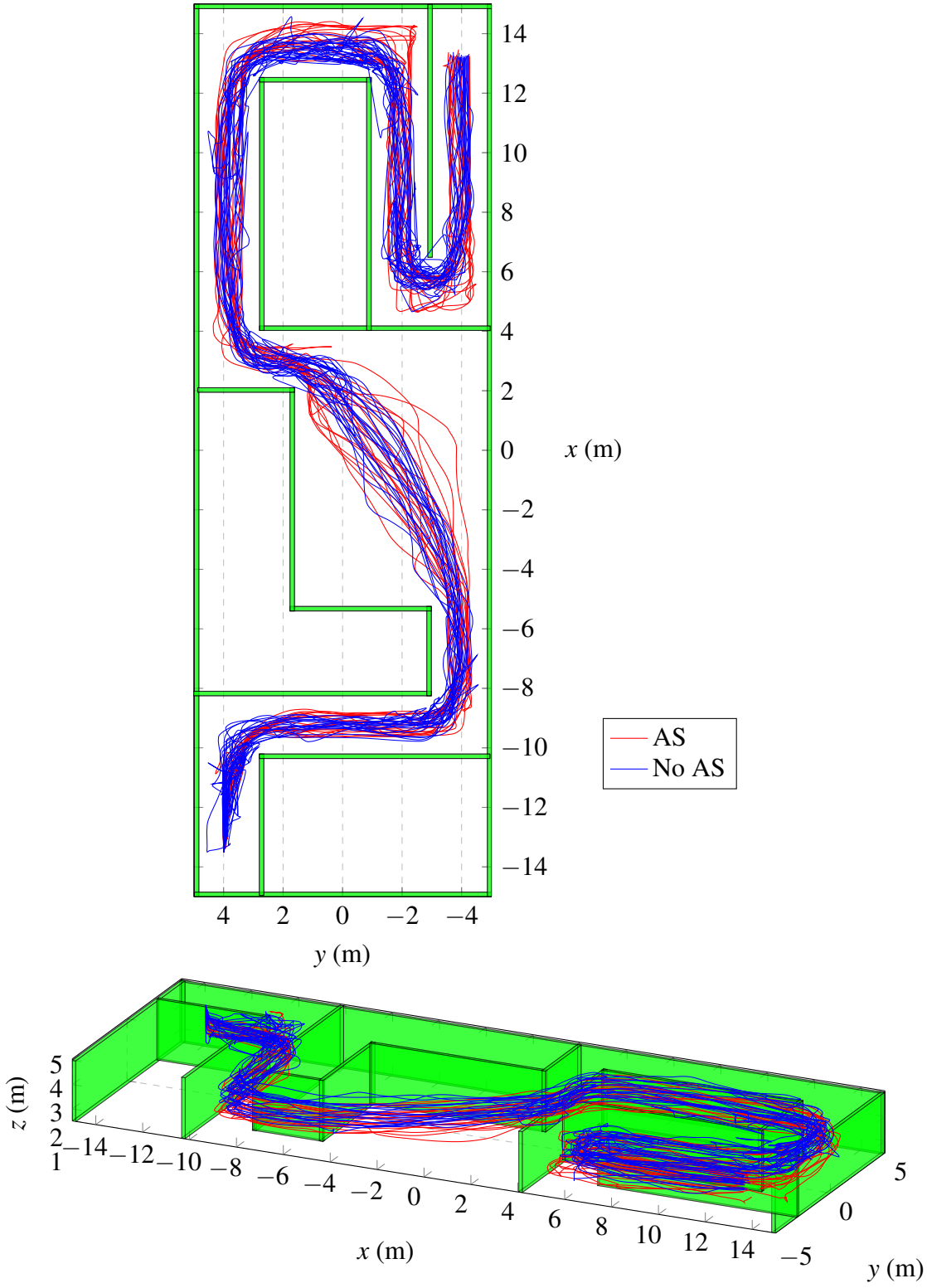


Figure 5.23: Trajectories of the participants in the first scenario from two different views. The trajectories are sub sampled by a factor of 50. Each participant has done two runs on this figure, one in red and one in blue

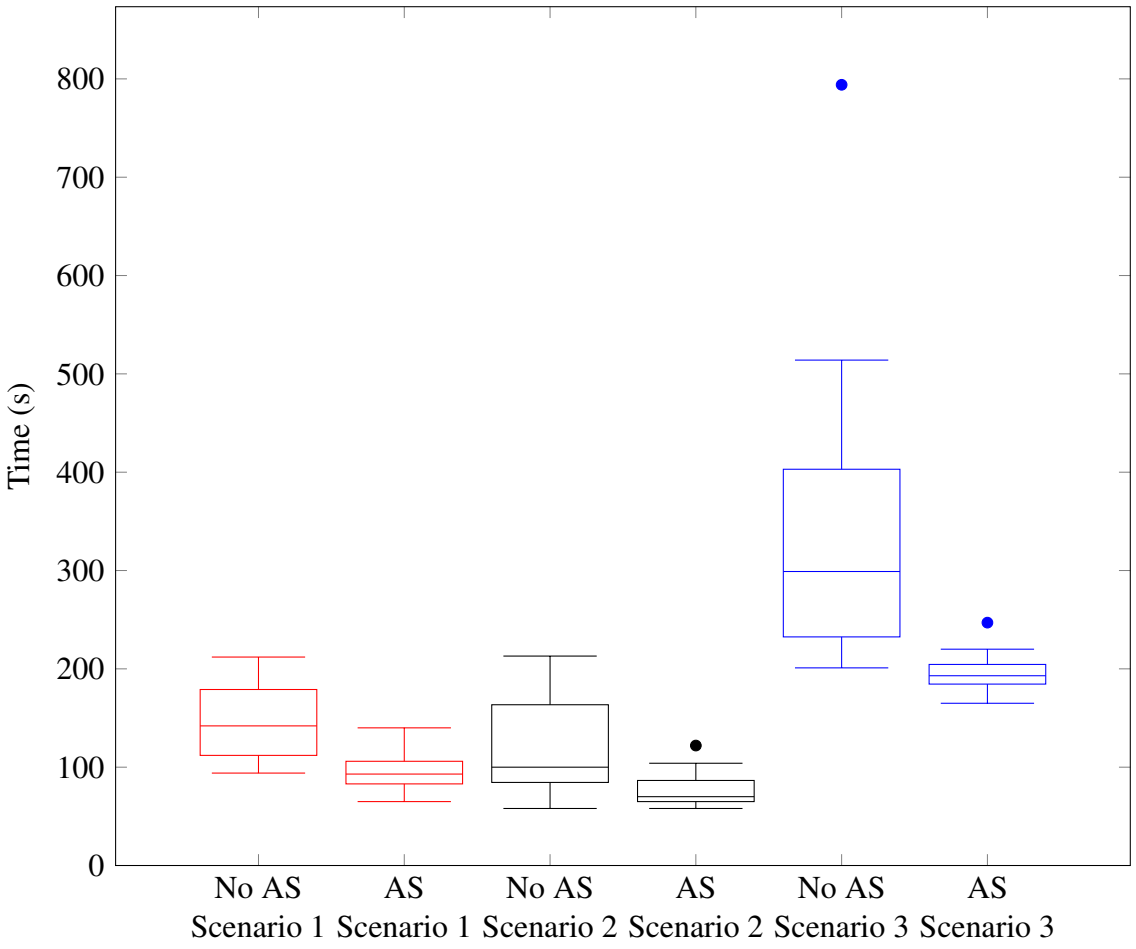


Figure 5.24: Time needed to complete the task with and without the avoidance system

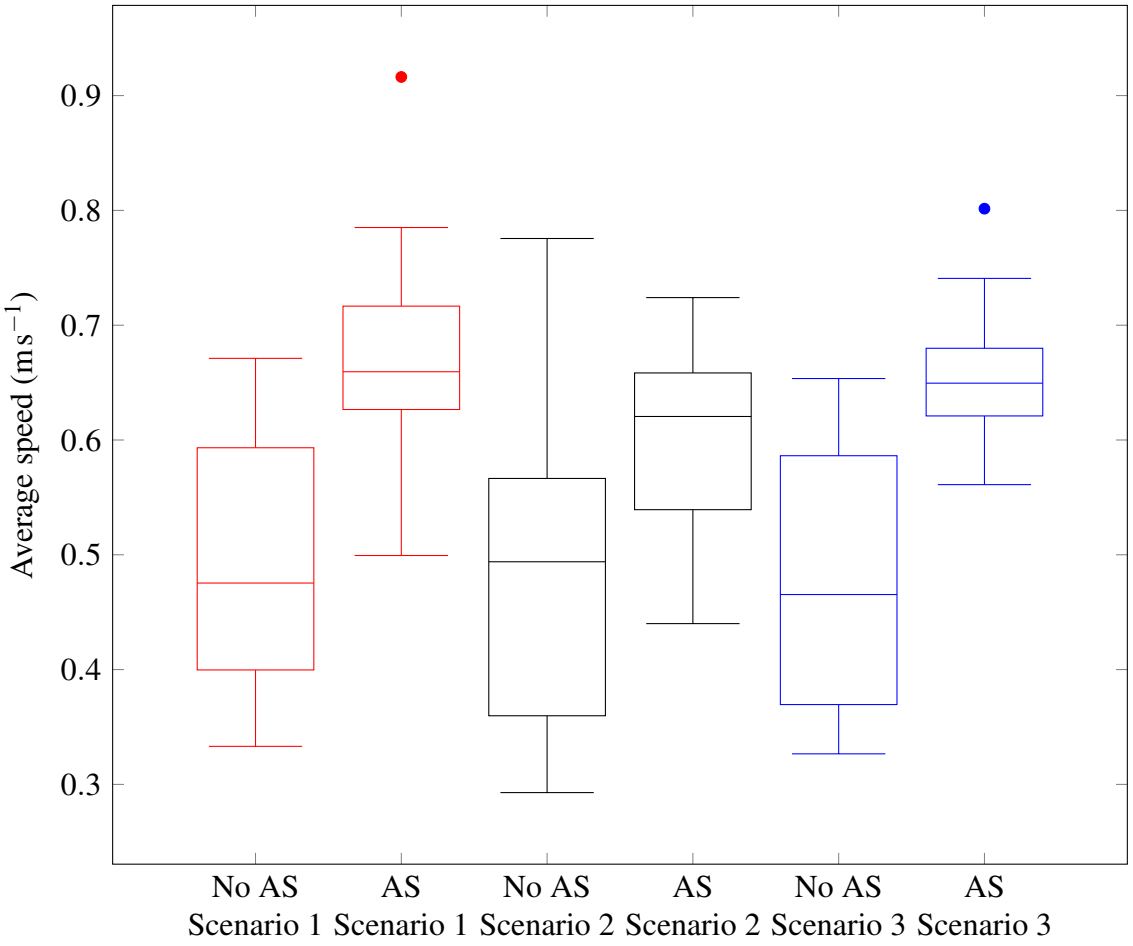


Figure 5.25: Average speed of the UAV during the task with and without the avoidance system

to complete the task in all three scenarios. This reduction is statistically significant for the three scenarios (p-values for scenarios 1,2 and 3 are respectively 0.0001, 0.0008 and < 0.0001) and amounts to 39 %, 40 % and 45 % for scenarios 1, 2 and 3 respectively. Moreover, the variance of this duration is reduced as well for all scenarios as seen on Figure 5.24. This suggests that the obstacle avoidance algorithm negates the differences in difficulty of the scenario and skills of the user. A correlated metric, the average speed of the UAV, is shown in Figure 5.25. The speed of the UAV was averaged over the whole scenario per scenario and per person. The results are used to create the box plots, with 20 data points per box plot. It is shown that the average speed of the UAV is higher when using the obstacle avoidance scheme, with again a smaller variance of this average speed between users. The difference made by the algorithm is statistically significant as shown

in Table 5.2 and amounts to an increase in average speed of 31 %, 35 % and 32 % for scenario 1, 2 and 3 respectively.

5.7.3 Workload : NASA TLX results

The results of the NASA TLX evaluation are provided in Figure 5.26. The full test (including pairwise comparison of the factors) was performed. For the reasons mentioned in Section 2.4.3, the weighted and non weighted results are plotted. However, the trend exhibited by the results is the same regardless of the weighting method, so the analysis is limited here to the weighted results. The workload is shown to be reduced by the obstacle avoidance algorithm and this reduction is statistically significant, from 56.5 points to 26.5 points of NASA TLX score.

5.7.4 Conclusions from the experiment

These experiments show that the proposed approach improves safety by reducing collisions, while giving to the operator a liberty of movement since the objective is not to maximise the distance to an obstacle. OAST also reduces the time required to complete the scenarios. Those elements are consistent with the reduction of workload that is observed with the algorithm. Indeed, considering some of the NASA TLX metrics: a reduction of the number of collisions is likely to reduce frustration, a safety net might reduce mental demand, a liberty of movement is expected to not increase the physical demand (since the user does not have to fight against the joystick) and a lower time of completion is likely to improve the performance feeling.

It is interesting to note that the proposed algorithm seems to suppress the difference of skills between the participants: the number of collisions and the time needed to complete the scenarios have a reduced variance compared to runs performed without the algorithm. A possible explanation is that the advantages of OAST are greater when the skill of the operator is lower or the situation is more difficult to handle. Considering the number of collisions and the time elapsed, the difference made by the algorithm is more important

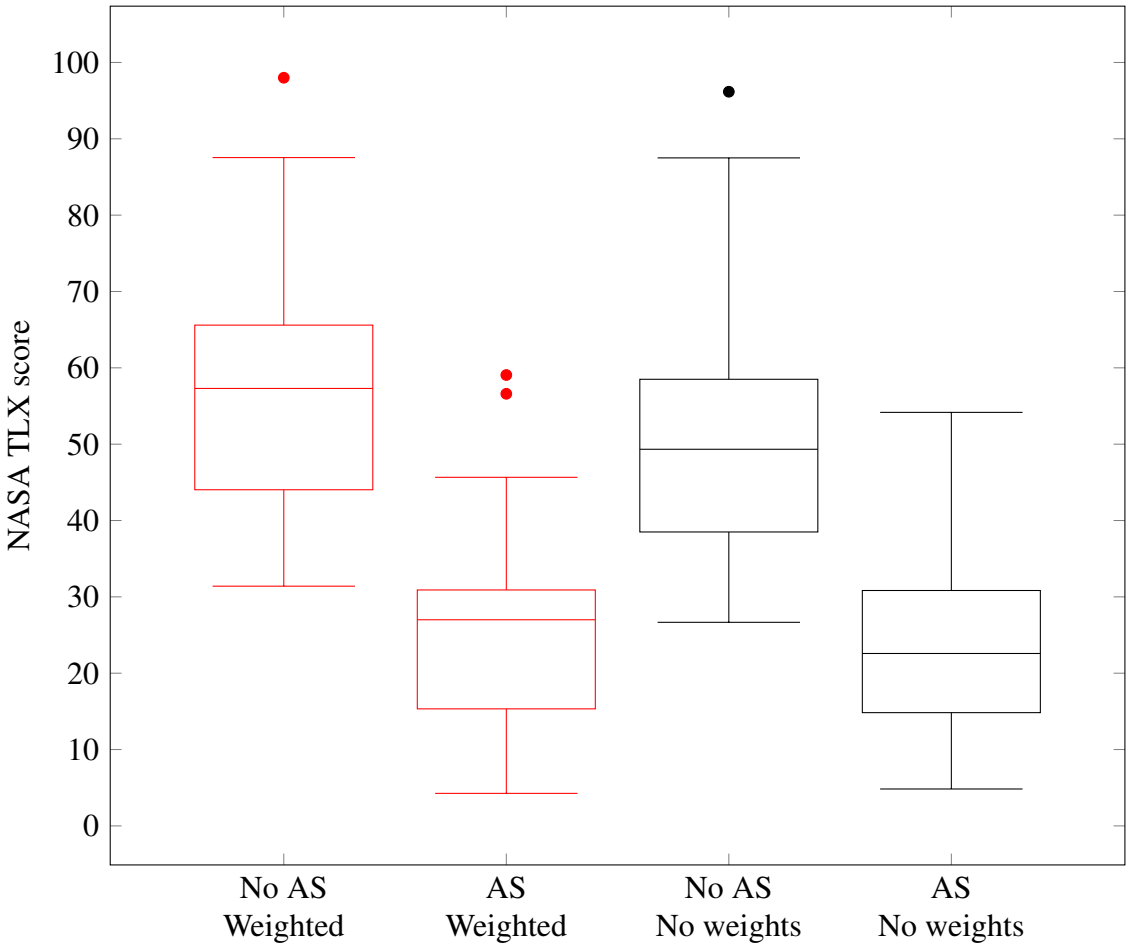


Figure 5.26: NASA TLX final workload score with and without the obstacle avoidance system. The weighted and non weighted results are presented

in scenario 3 than in the simpler scenario 2.

These experiments cannot be compared quantitatively to the experiments performed in Section 2.3 since the protocol is different. It is however possible to draw qualitative conclusions. In Chapter 2, the implemented scheme does not override the human input. It influences this input by steering the end effector in a desired direction. In this chapter, the human input is overridden and a haptic feedback informs the user of the nature of the override. In both cases, the challenge is to design a method that allows the operator to keep a freedom of movement. This constraint can translate to a property of transparency for the algorithm: if the user does not notice the algorithm then a workload increase becomes less likely. Based on the subjective observation of the participants, the algorithm developed in this chapter, although it modifies the user input, appears less intrusive than the one presented in Chapter 2. The absence of a strong physical feedback means that users do not struggle against the joystick and objective metrics such as time of completion and workload suggest that the users were able to adopt the trajectory they wanted.

5.8 Considerations due to latency

Since operators are physically separated from the UAVs during teleoperation, the latency of communication between the ground station and the UAV is a concern. In this section, the implications of latency are discussed. In particular, its influence on the maximum speed of the UAV is examined. The influence on the stability of the teleoperation scheme is then briefly discussed.

5.8.1 Velocity limit

This thesis focuses on indoor UAV teleoperation, meaning that obstacles are expected to be numerous and generally close to the UAV. For this reason, the maximal velocity of the UAV is expected to be low. The developed algorithms are not designed for fast drones in applications such as racing. The capacity of the obstacle algorithm to perform its task

efficiently depends on several variables, the first one being the latency of the algorithm. This latency depends on the sensor as well as the processing time, which itself depends on the algorithm and hardware used. The second variable is the agility of the UAV, which can be quantified by the maximum possible acceleration of the UAV in a given direction. The third variable is the obstacle itself, its location around the UAV and its size. The general question of the maximum speed of a UAV depending on latency, agility of the UAV and obstacle size has been studied in the work of Falanga, Kim, and Scaramuzza [139]. In this paper, the robot is moving along the longitudinal axis with a fixed speed, and can avoid the obstacle in the lateral direction with an acceleration a_m^l. The obstacle is considered to be a cube, whose size is increased by the radius of the robot for a total radius r_o. An obstacle is detected when it enters the sensing range s. The time between the detection of an obstacle and the initiation of an avoidance maneuver is τ. The maximum longitudinal speed v_m that the robot can achieve to avoid the obstacle is shown by the authors to be [139]:

$$v_m = \frac{s}{\tau + 2\sqrt{\frac{r_o}{a_m^l}}}. \tag{5.39}$$

The authors explain that the value of a_m^l depend on the thrust-to-weight ratio, with commercially available UAVs ranging from $10\,\mathrm{m\,s^{-2}}$ (thrust-to-weight ratio of approximately 1.5) to $50\,\mathrm{m\,s^{-2}}$ (thrust-to-weight ratio of approximately 5.2) [139]. The thrust-to-weight ratio of the Pelican, the UAV considered in our experiments, varies between 2.22 and 3.66 depending on the payload. It is thus conservative to consider a maximal lateral acceleration of $10\,\mathrm{m\,s^{-2}}$ for this UAV. Another way to look at the lateral acceleration is to consider the necessary steady state roll angle using the standard dynamic model of a quadrotor, e.g., as developed in the work of Bouabdallah and Siegwart [136]. From this model, using a maximal thrust of 36 N for the UAV and a weight of 1.65 kg, an acceleration of $10\,\mathrm{m\,s^{-2}}$ would require a roll angle of around 27°, which is below the maximal roll angle of 63° to keep the same height (which would result in an acceleration of around $19\,\mathrm{m\,s^{-2}}$). This reasoning does not consider the time to achieve this angle, since a steady

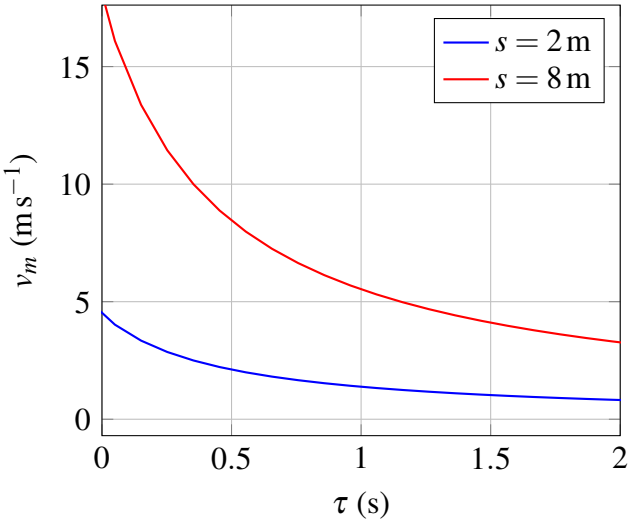


Figure 5.27: Maximum longitudinal speed of the UAV in order to laterally avoid an obstacle of size $r_o = 0.5\,\mathrm{m}$

state is assumed, but it shows that the considered value is consistent with the thrust-to-weight ratio of this UAV. The sensing range s is dependent on the sensors used, for a stereo camera, a value between 2 m and 8 m is reasonable [139]. While a LIDAR sensor has a higher range, typically more than 30 m, in cluttered environments, obstacle are closer to the UAV than 30 m, so the sensing range considered is between 2 m and 8 m. The size r_o of the obstacle is fixed to 0.5 m. The maximum longitudinal speed available to the UAV in order to laterally avoid the obstacle is illustrated in Figure 5.27, computed from equation (5.39). The scenario considered above is a lateral obstacle avoidance, with no modification of the longitudinal speed. If the goal is rather to stop before hitting the obstacle, the computation is different. Using the same notations, the distance before stopping is the sum of the distance crossed during τ at the initial speed v of the UAV and the distance necessary to stop from a speed v by applying a deceleration a_m^f. In other word, the minimal stopping distance is:

$$v\tau + \frac{v^2}{2a_m^f}. \tag{5.40}$$

The collision occurs when this distance is shorter than the sensing distance s, thus the maximum longitudinal speed v_m available to the UAV in order to stop before an obstacle

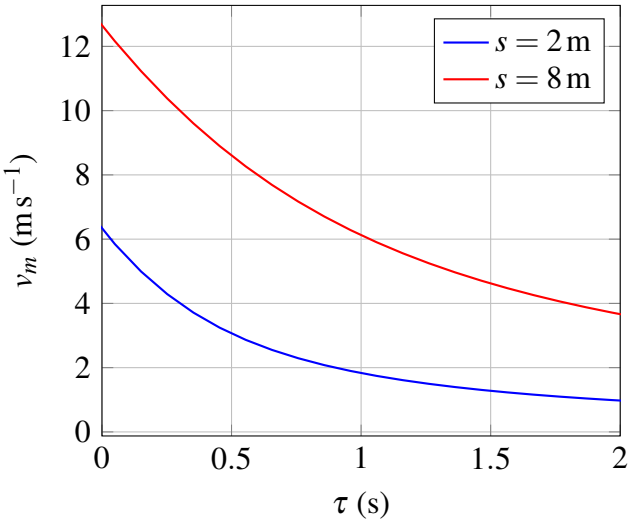


Figure 5.28: Maximum longitudinal speed of the UAV in order to stop before an obstacle

is:

$$v_m = -a_m^f \tau + \sqrt{a_m^{f2} \tau^2 + 2 a_m^f s}. \tag{5.41}$$

The same values are kept for the parameters involved in the computation, although it is important to mention that while the maximum steady state value of the acceleration is the same in the lateral and longitudinal direction, the time necessary to reach this value is longer along the longitudinal axis, since the pitch angle starts at a positive value. The maximum longitudinal speed available to the UAV in order to stop before the obstacle is illustrated in Figure 5.28, computed from equation (5.41). The latency depends on the sensor and hardware. Moreover, it is important to differentiate between the two types of obstacle avoidance algorithms that have been developed in this thesis. The first one, subject of Chapter 2, relies on the haptic feedback generated by the joystick to avoid obstacle. This means that, once an obstacle is detected, and the feedback computed, this feedback has to be sent to the joystick. Then, the modification of the trajectory due to the change of the position of the end effector has to be sent back to the UAV. This means that for this algorithm, the latency of the data link between the ground station and the UAV impacts the performances of the algorithm. In the second case, which is the OAST

algorithm, the trajectory is directly modified on board, thus the data link does not add any latency.

The sensor latency changes depending on the type of sensor considered, with a stereo camera, an upper bound can be chosen around 70 ms [139], while the LIDAR sensors considered in this thesis acquire a scan in around 100 ms. Thus, a sensor latency of 100 ms is considered, which is the worst case scenario. On the hardware used in this thesis, the runtime of both obstacle avoidance algorithms is low, typically less than 20 ms. However, in the case of the OAST algorithm, a map needs to be computed. Using only the registration of LIDAR scans to compute the odometry results in a runtime of around 200 ms, which is highly dependent on the mapping parameters, as detailed in section 3.5. It is important to repeat that, in a realistic setting, several odometry sources would be fused (e.g., IMU, cameras) in order to get a faster computation. Nonetheless, in order to consider a worst case scenario, a computation time of 200 ms is considered for the mapping process of OAST. Finally, as explained above, the data link latency needs to be considered for the algorithm described in Chapter 2. A study of the delay incurred by data links when communicating with UAVs [140] gives an estimation of the latency between 110 ms and 1110 ms for line-of-sight data links, and between 670 ms and 1670 ms for geostationary satellites relay. Given the domain of application of this thesis (e.g., inspection, search and rescue), line-of-sight data links are a reasonable assumption. Summing those delays, the total latency of the OAST algorithm is approximately 320 ms, while the total latency of the algorithm proposed in Chapter 2 varies between 230 ms and 1230 ms depending on the data link.

Using Figure 5.27 and Figure 5.28, the OAST algorithm allows a maximum speed of the UAV of $2.6\,\mathrm{m\,s^{-1}}$ and $10\,\mathrm{m\,s^{-1}}$ for a sensing range of 2 m and 8 m respectively for a lateral avoidance, and $3.9\,\mathrm{m\,s^{-1}}$ and $9.8\,\mathrm{m\,s^{-1}}$ for a sensing range of 2 m and 8 m respectively for stopping before the obstacle. It has to be noted that, by design, the OAST algorithm is able to perform lateral avoidance maneuvers, since the direction of travel will often result in sliding along an obstacle, as described in section 5.4. The advantage

of such a maneuver compared to a stop before the obstacle is that the UAV conserves its longitudinal speed, saving time and allowing a smoother teleoperation.

On the other hand, for the algorithm proposed in Chapter 2, in the case of lateral avoidance, the maximum speed of the UAV varies between $3.0\,\mathrm{m\,s^{-1}}$ and $1.2\,\mathrm{m\,s^{-1}}$ for a sensing range of 2 m and between $12.0\,\mathrm{m\,s^{-1}}$ and $4.8\,\mathrm{m\,s^{-1}}$ for a sensing range of 8 m. Considering the maneuver to stop before an obstacle, the range of those speeds becomes between $4.4\,\mathrm{m\,s^{-1}}$ and $1.5\,\mathrm{m\,s^{-1}}$ for a sensing range of 2 m and between $10.5\,\mathrm{m\,s^{-1}}$ and $5.3\,\mathrm{m\,s^{-1}}$ for a sensing range of 8 m.

5.8.2 Stability of the teleoperation scheme

In addition to the limitation of the maximum speed of the UAV, the latency in the teleoperation pipeline has heavy implications on the stability of the teleoperation scheme. While a thorough treatment of this topic is outside the scope of this thesis, this section aims to provide some information and potential solutions to the stability problem.

There are two different issues when considering latency in teleoperation: the latency of the visual feedback and the latency of the haptic feedback. The latency of the visual feedback causes problems of efficiency, but not of stability [141]. The strategy adopted by the operators to cope with the time delay is called ‘move-and-wait’: the operator performs a small movement of the robot, then waits to observe the result of this movement on the screen before moving again. This method then slows down the teleoperation, reducing the efficiency.

However, during bilateral teleoperation, the delay of the force feedback causes stability issues [142], even with small time delays [143]. Thus, while the haptic feedback might improve the safety of teleoperation, as discussed in Chapter 2, it can introduce stability issues as well. It is important to remind the difference between the obstacle avoidance algorithms presented in this thesis. In Chapter 2, the proposed algorithm relies exclusively on the haptic feedback to perform obstacle avoidance. This means that stability issues due to time delays are a major concern. In the case of OAST however, the haptic feedback

is optional, and used to facilitate the interaction between the operator and the obstacle avoidance system. As such, it is possible to disable the haptic feedback completely without any safety downside, although there might be negative consequences on the efficiency and operator workload. As a result, since the haptic feedback is optional and lighter (i.e. the forces used are less strong) in the case of OAST, considerations of stability are less of a concern for OAST, which is the main result of this thesis. The following is thus mainly relevant for the algorithm presented in Chapter 2.

In order to preserve the stability of the teleoperation system independently of time delays, the scattering method was proposed to preserve passivity [144]. This theory was reformulated by introducing the notion of 'wave variables' [145] for master slaves systems. The evolution of this concept has been described in the work of Niemeyer and Slotine [146]. The principle is to apply a bijective transformation to the velocity $\dot{\mathbf{x}}$ and the force $\mathbf{F}$ to obtain two new so called waved variables $\mathbf{u}$ and $\mathbf{v}$ by:

$$\mathbf{u} = \frac{b\dot{\mathbf{x}} + \mathbf{F}}{\sqrt{2b}}, \tag{5.42}$$

$$\mathbf{v} = \frac{b\dot{\mathbf{x}} - \mathbf{F}}{\sqrt{2b}}, \tag{5.43}$$

where b is a positive tuning constant. $\mathbf{u}$ is called the forward moving wave and $\mathbf{v}$ the backward moving wave. While the wave theory does not precise if the master sends a force or a velocity, in our case, a velocity or position would be sent by the master to the slave and a force feedback would be sent to the master by the slave. The idea of the wave theory is that $\dot{\mathbf{x}}$ and $\mathbf{F}$ are not exchanged directly, but first converted into $\mathbf{u}$ and $\mathbf{v}$. Then, the only variables exchanged between the master and the slave are $\mathbf{u}$ and $\mathbf{v}$, from which $\dot{\mathbf{x}}$ and $\mathbf{F}$ can be retrieved due to the bijective nature of the transformation. It is shown that doing so ensures a passive and lossless communication [146], independently of the time delay in the system.

The wave theory has been adapted to a teleoperated UAV [147], where the position

of the master represents a position or velocity command for the slave, and virtual forces are sent back to the master (computed from a force field or a map). This study shows that when using the haptic feedback to help with obstacle avoidance, which is the approach adopted in Chapter 2, the presence of time delays increases the number of collisions and the workload of the operator [147]. Applying the wave variables then significantly reduced the number of collisions and operator workload. The authors explain that the operators perceived a more reactive system since, in the presence of an obstacle, the UAV starts to decelerate before reception of the haptic feedback by the operators. This means that the resulting system is now hybrid, since the UAV is affected both by the commands of the operator and the force field computations. To summarise, the wave variables can be applied to a teleoperated UAV relying on haptic feedback for avoiding obstacles in presence of time delays, which mitigates the performance loss caused by time delays.

In the OAST algorithm, the haptic feedback only has an indicative role, as mentioned above, thus it would be interesting to examine the consequences of time delays for this algorithm. Should any negative consequence arise, the application of the theory of wave variables could be a future avenue of research.

5.9 Conclusion

In this chapter, a complete flight assistance assistance system for intelligent teleoperation is proposed. This system builds on the map developed in Chapter 3. The proposed approach modifies the user input to increase the safety of the trajectory. This modification is based on the original input in order to keep a final trajectory close to the user's input. The algorithm is extensively tested in a simulated environment through a human experiment involving inexperienced operators. The approach is shown to improve both safety and efficiency, while reducing the workload.

Chapter 6

Conclusions and future work

Contents

The main objective of this thesis is to develop a flight assistance system to make the teleoperation of small rotary wings UAVs safer and more efficient. Such a system involves elements of human-machine interaction as well as obstacle detection, representation and avoidance. This work begins in Chapter 2 with a method of obstacle avoidance using a haptic feedback provided by the haptic controller used to operate the drone. The obstacle representation is then improved through the use of a mobile map in Chapter 3 and an algorithm fusing the output of stereo cameras and LIDARs in Chapter 4. With those improvements in perception, a new flight assistance system is proposed in Chapter 5, using the map to prevent collisions from occurring and increasing efficiency.

In this chapter, the main findings of this thesis are summarised as well as limitations and avenues of improvement through future work.

6.1 Haptic feedback for obstacle avoidance

A 3D haptic controller is used throughout this thesis to send control commands to the UAV, for two reasons. Firstly, the 3D haptic controller is simpler to operate than a standard remote controller since the movement of the end effector is directly related to the movement of the UAV. With a standard remote controller, the thrust, roll, pitch and yaw are controlled separately. Secondly, the haptic feedback can be used to improve the human-machine interface. The haptic feedback is used in Chapter 2 as the main way to perform obstacle avoidance and in Chapter 5 as an information given to the operator about the behaviour of the obstacle avoidance scheme.

In Chapter 2, the haptic feedback steers the end effector of the haptic controller away from the position of the detected obstacles, which in turn repels the UAV away from the obstacles. Since this method is based on potential fields to compute the force feedback, it is susceptible to local minima. In presence of close obstacles, this translates to situations where it is difficult to manoeuvre the UAV due to the force feedback. A hole detection algorithm is implemented to alleviate this problem for obstacles of symmetric shape. Since this method is designed for teleoperation, it is tested in three closed virtual environments by untrained human operators. Those environments contain narrow corridors and windows, which make them challenging to navigate through. This experiment shows that the haptic feedback implemented in the proposed algorithm increases safety in one of the scenario. The hole detection algorithm allows a reduction in workload for the operator at a safety cost. Overall, the experiment suggests that while the haptic feedback makes teleoperation safer, it also reduces the manoeuvrability of the UAV. It is then proposed to perform automated obstacle avoidance, while using the haptic feedback to inform the operator about the decisions of the algorithm. This method is ultimately described in Chapter 5.

However, such a method requires a precise representation of the obstacles. In Chapter 2, the input is a single LIDAR scan, which lacks density. In order to have a better representation of the obstacles, two methods are presented: the accumulation of several

point clouds into a mobile map is described in Chapter 3 and the additional usage of stereo cameras to increase the density of LIDAR scans is proposed in Chapter 4.

6.2 Perception and representation of obstacles

The first method to get a more precise representation of the environment is to accumulate the point clouds over time, i.e. to create a map of the environment. This map however requires the odometry of the robot to be computed. The NDT OM is a framework that allows both the creation of a map and the computation of the robot's odometry from this map. The generated map is however fixed in space. The NDT RC algorithm is developed to allow this map to move with the robot. Improvement is also brought to the raytracing process of the NDT OM implementation. When applied to the testing part of the KITTI odometry dataset, NDT RC performs well, with a translation error of 1.54 cm and a rotation error of 0.106 mrad. In the Ford dataset, a mean absolute trajectory error of 1.42 m is obtained. The runtime of the mapping algorithm is 187 ms and 229 ms on the KITTI and Ford datasets respectively, making NDT RC suitable for real time use.

A limitation of this method is the high number of parameters, which can make the tuning challenging given that the parameters are correlated (e.g., the cell size, α, β, γ, M_{max}, K_{occ}, defined in Chapter 3). A tuning method can be developed to efficiently set those parameters for a particular sensor and environment.

A second possibility of future work is the combination of the mobile algorithm developed in this thesis with a more efficient NDT matching algorithm to improve odometry. Indeed, the mobile NDT allows improvement to the odometry by accumulating the scans. On the other hand, some algorithms (e.g., the semantic assisted NDT [91]) improve the odometry by working on the matching stage. It is therefore possible to combine both approaches in order to get a more precise and faster odometry computation.

The second method employed to improve the obstacle perception is to fuse the output of the LIDAR with the one from a stereo camera. Indeed, while LIDARs offer precise 3D

points, their output is sparse, especially for models that are susceptible to be mounted on a small UAV. This can make the detection of small obstacles challenging. In Chapter 4, an algorithm is developed to increase the density of point clouds from a LIDAR by fusing them with images from stereo cameras. The accuracy of the resulting depth map is evaluated on the KITTI stereo dataset. The proposed method achieves 4.17 % of erroneous pixels and a RMSE of 3.59 pixels, which compares favourably with other fusion methods. Moreover, the influence on this fusion scheme on the mapping process is evaluated.

Qualitatively, it is found that adding the stereo data would provide a more precise representation for some small obstacles. The benefit provided by the stereo fusion increases when the density of the LIDAR scan decreases. In order to get a quantitative analysis, the KITTI odometry dataset is used to evaluate the influence on the fusion scheme for two different LIDAR types. The addition of the full depth map deteriorates the odometry, however, limiting the depth of the points coming from the depth map allows the odometry to be approximately the same, while still benefiting from the increase in density. Should the density of the LIDAR decrease, then the addition of the stereo data provides a precision benefit.

A limitation of this method is the runtime, which increases by approximately 400 ms. The algorithm is however well suited for a GPU implementation, which could be a way to significantly speed up the method. The recent development of embedded GPUs would make this solution suitable for small UAVs.

6.3 Flight assistance system for UAV teleoperation

The availability of the map allows the design of an obstacle avoidance algorithm that actively modifies the input from the operator. The complete system, OAST, is described in Chapter 5, from the input of the operator through the 3D haptic controller to the modification of this input. In accordance with the conclusions of Chapter 2, OAST does not only aim at avoiding crashes but also at assisting the operator to reach the desired posi-

tion goal. The role of the haptic feedback is now indicative: it informs the operator on the modification of the input performed by the algorithm. OAST is tested in a computerised experiment involving 20 subjects, all inexperienced with UAV teleoperation. It is shown that the flight assistance system significantly improves safety: almost no collision was recorded when the algorithm was used. Moreover, the temporal efficiency improved as well with the algorithm, with a reduction of the time needed to complete a scenario between 39 % and 45 % depending on the scenario. Finally, the workload of the operator, as measured by the NASA TLX test [41], was reduced. It is important to note that this experiment was performed with non trained people. Reducing the difficulty of teleoperation of UAVs could lead to a broader usage of those drones.

While OAST shows promising results, it was evaluated using the ground truth odometry. The fusion of the odometry computed from the map (considering the odometry improvements suggested in the section above) with an IMU and the cameras should allow a precise odometry to be computed in real time.

A second suggestion for future work is to use a haptic controller in the admittance framework. This thesis has been conducted with haptic controllers working in impedance mode, meaning that the operator applies a displacement to the end effector and a force feedback is received. In the admittance framework, the situation is reversed: the user applies a force to the end effector and a displacement feedback is received. The improvement possibilities by integrating admittance framework to the proposed method can be investigated.

A third interesting question is the influence of time delays on the OAST algorithm. A potential solution to stability issues arising from those time delays could be the wave variables theory.

A final direction for additional research is the implementation on a real UAV. The code structure could be used directly onboard, however the sensors would need to be mounted and powered.

Appendix A

NASA Task Load Index (TLX) form

The paper version of the NASA TLX form is presented in Figure A.1 without the cards for pairwise comparisons[1]. There are six different scale. For each of them, the participant is asked to carefully place a point on the scale, generating a score between 0 and 100 per scale.

The second part is the pairwise comparison, that aims to find which scales contributed the most to the workload. It is for instance possible that a user would find a task physically demanding, although the physical demand did not contribute a lot to the workload compared to the other factors. There are 6 scales, thus 15 possible pairs of scales. The user is presented with those 15 pairs and for each pair, has to select the scale that contributed the most to the workload. Each selected scale gains 1 point. A weight is created for each scale by dividing its number of points by 15. The final score is computed with a weighted sum of the scale results by their weights.

[1] Source: `https://humansystems.arc.nasa.gov/groups/TLX/downloads/TLXScale.pdf`

Figure 8.6

NASA Task Load Index

Hart and Staveland's NASA Task Load Index (TLX) method assesses work load on five 7-point scales. Increments of high, medium and low estimates for each point result in 21 gradations on the scales.

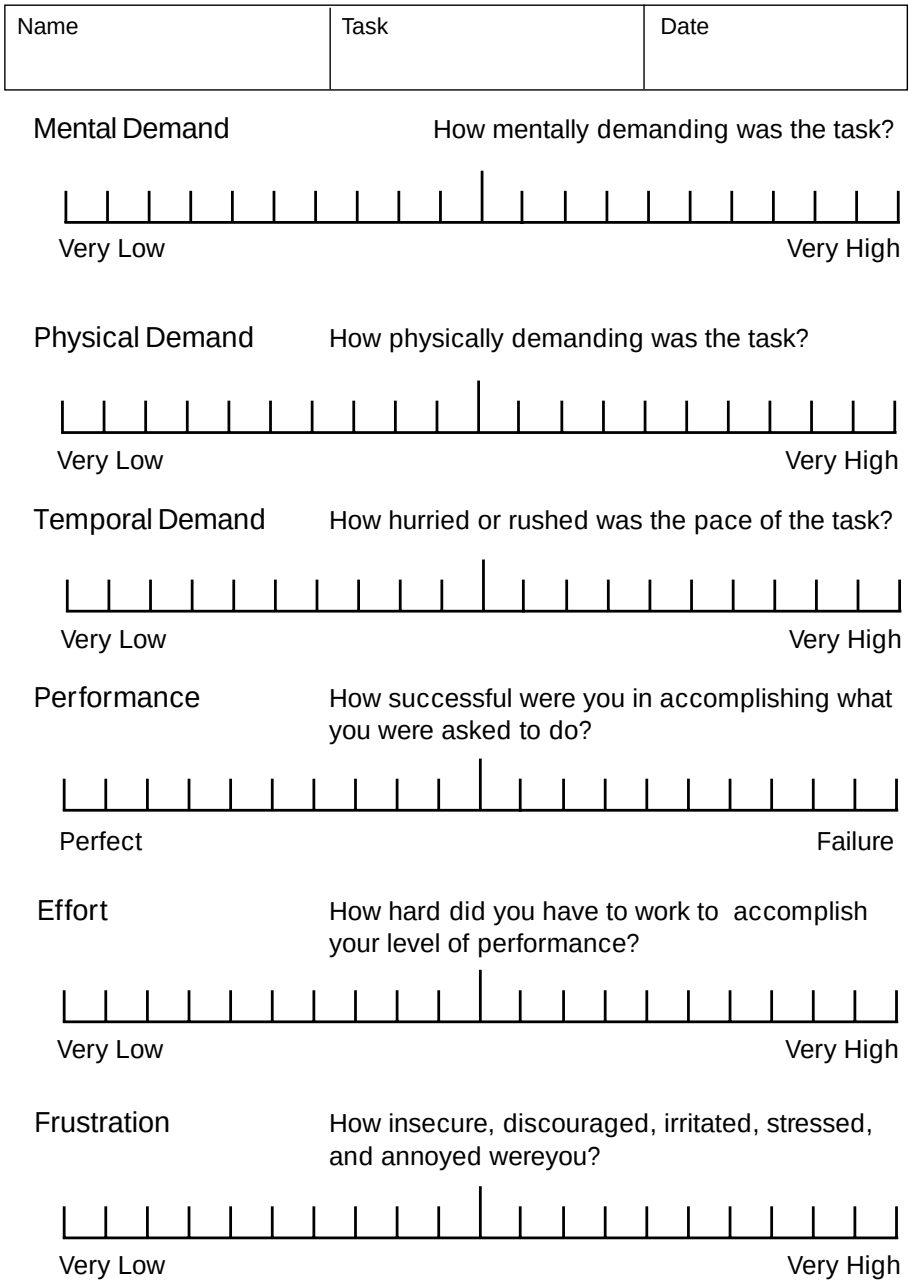


Figure A.1: The paper version of the NASA TLX without pairwise comparison of the scales.

Appendix B

Proofs

B.1 Product of normal probability density functions

Let $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})$ be the probability density function of a multivariate Gaussian distribution in a n dimensional space, with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Let $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ be two mean vectors and $\boldsymbol{\Sigma}_i$ and $\boldsymbol{\Sigma}_j$ be two positive definite covariance matrices.

The objective is to prove that the following holds:

$$\int_{\mathbb{R}^n} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)\mathbf{dx} = \mathcal{N}(\mathbf{0}|\boldsymbol{\mu}_i-\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j).$$

We note $p = \int_{\mathbb{R}^n} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j)\mathbf{dx}$.

A first intermediate result is the following, for a symmetric positive definite matrix $\mathbf{A}$ and a vector $\mathbf{j}$:

$$\int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{x}^t\mathbf{A}\mathbf{x}+\mathbf{j}^t\mathbf{x}}\mathbf{dx} = \sqrt{\frac{(2\pi)^n}{\det(\mathbf{A})}}e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}}. \tag{B.1}$$

In order to prove this, a change of variable $\mathbf{z} = \mathbf{x}-\mathbf{A}^{-1}\mathbf{j}$ is performed:

$$\int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{x}^t\mathbf{A}\mathbf{x}+\mathbf{j}^t\mathbf{x}}\mathbf{dx} = \int_{\mathbb{R}^n} e^{-\frac{1}{2}\left(\mathbf{z}^t\mathbf{A}\mathbf{z}+\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}+2\mathbf{j}^t\mathbf{z}\right)+\mathbf{j}^t\mathbf{z}+\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}}\mathbf{dz} \tag{B.2}$$

$$= e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}}\int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{z}^t\mathbf{A}\mathbf{z}}\mathbf{dz}. \tag{B.3}$$

Then, $\mathbf{A}$ is diagonalised: since $\mathbf{A}$ is real symmetric, there exists $\mathbf{Q} \in \mathrm{O}(n)$ and a diagonal matrix $\mathbf{D} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ so that $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^t$. A second change of variable is performed: $\mathbf{l} = \mathbf{Q}^t\mathbf{z}$. The absolute value of the determinant of the Jacobian of this transformation is 1, thus injecting this into the integral yields:

$$\int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{x}^t\mathbf{A}\mathbf{x}+\mathbf{j}^t\mathbf{x}}\mathbf{dx} = e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}} \int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{l}^t\mathbf{D}\mathbf{l}}\mathbf{dl} \tag{B.4}$$

$$= e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}} \prod_{i=1}^{n} \int_{\mathbb{R}^n} e^{-\frac{1}{2}\lambda_i l_i^2}\mathrm{dz_i} \tag{B.5}$$

$$= e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}} \prod_{i=1}^{n} \sqrt{\frac{2\pi}{\lambda_i}} \quad \text{using the Gaussian integral} \tag{B.6}$$

$$= e^{\frac{1}{2}\mathbf{j}^t\mathbf{A}^{-1}\mathbf{j}} \sqrt{\frac{(2\pi)^n}{\det(\mathbf{A})}}, \tag{B.7}$$

which proves equation (B.1).

Developing the integral p yields:

$$p = \frac{1}{(2\pi)^n \sqrt{\det(\boldsymbol{\Sigma}_i)\det(\boldsymbol{\Sigma}_j)}} \int_{\mathbb{R}^n} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)}\mathbf{dx}. \tag{B.8}$$

A change of variable $\mathbf{y} = \mathbf{x} - \boldsymbol{\mu}_i$ gives:

$$p = \frac{1}{(2\pi)^n \sqrt{\det(\boldsymbol{\Sigma}_i)\det(\boldsymbol{\Sigma}_j)}} \int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{y}^t\boldsymbol{\Sigma}_i^{-1}\mathbf{y}-\frac{1}{2}(\mathbf{y}+\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}+\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}\mathbf{dy} \tag{B.9}$$

$$= \frac{e^{-\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}}{(2\pi)^n \sqrt{\det(\boldsymbol{\Sigma}_i)\det(\boldsymbol{\Sigma}_j)}} \int_{\mathbb{R}^n} e^{-\frac{1}{2}\mathbf{y}^t(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1})\mathbf{y}-(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}\mathbf{y}}\mathbf{dy}. \tag{B.10}$$

Equation (B.1) is used to compute the integral since $\boldsymbol{\Sigma}_i^{-1} + \boldsymbol{\Sigma}_j^{-1}$ is symmetric positive definite:

$$p = \frac{e^{-\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}}{(2\pi)^n \sqrt{\det(\boldsymbol{\Sigma}_i)\det(\boldsymbol{\Sigma}_j)}} \sqrt{\frac{(2\pi)^n}{\det(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1})}} e^{\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1})^{-1}\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}. \tag{B.11}$$

The denominator can be simplified:

$$\det(\boldsymbol{\Sigma}_i)\det(\boldsymbol{\Sigma}_j)\det(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1}) = \det\left(\boldsymbol{\Sigma}_i\left(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1}\right)\boldsymbol{\Sigma}_j\right) \tag{B.12}$$

$$= \det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j). \tag{B.13}$$

Thus:

$$p = \frac{e^{-\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}}{\sqrt{(2\pi)^n\det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)}}e^{\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1}\right)^{-1}\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)} \tag{B.14}$$

$$= \frac{1}{\sqrt{(2\pi)^n\det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)}}e^{\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\left(-\boldsymbol{\Sigma}_j^{-1}+\boldsymbol{\Sigma}_j^{-1}\left(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1}\right)^{-1}\boldsymbol{\Sigma}_j^{-1}\right)(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)} \tag{B.15}$$

$$= \frac{1}{\sqrt{(2\pi)^n\det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)}}e^{\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\left(-\boldsymbol{\Sigma}_j^{-1}+\left(\boldsymbol{\Sigma}_j\left(\boldsymbol{\Sigma}_i^{-1}+\boldsymbol{\Sigma}_j^{-1}\right)\boldsymbol{\Sigma}_j\right)^{-1}\right)(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)} \tag{B.16}$$

$$= \frac{1}{\sqrt{(2\pi)^n\det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)}}e^{\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t\left(-\boldsymbol{\Sigma}_j^{-1}+\left(\boldsymbol{\Sigma}_j\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\Sigma}_j+\boldsymbol{\Sigma}_j\right)^{-1}\right)(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)}. \tag{B.17}$$

A variation of the Woodbury matrix identity is used:

$$\left(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j\right)^{-1} = \boldsymbol{\Sigma}_j^{-1} - \left(\boldsymbol{\Sigma}_j+\boldsymbol{\Sigma}_j\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\Sigma}_j\right)^{-1}. \tag{B.18}$$

Inserting this into the previous equation gives:

$$p = \frac{1}{\sqrt{(2\pi)^n\det(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)}}e^{-\frac{1}{2}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^t(\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j)^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)} \tag{B.19}$$

$$= \mathcal{N}(\mathbf{0}|\boldsymbol{\mu}_i-\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j), \tag{B.20}$$

which is the desired result.

B.2 Conditional probability for three events

Let X, Y, and Z be three events in a probability space, so that $P(X,Y) \neq 0$ and $P(X) \neq 0$. The objective is to prove that the following conditional probability holds:

$$P(Z|X,Y) = \frac{P(Y|Z,X)P(Z|X)}{P(Y|X)}.$$

By definition of conditional probabilities:

$$P(Z|X,Y) = \frac{P(Z,X,Y)}{P(X,Y)} \tag{B.21}$$

$$= \frac{P(Y|Z,X)P(Z,X)}{P(X,Y)} \tag{B.22}$$

$$= \frac{P(Y|Z,X)P(Z|X)P(X)}{P(Y|X)P(X)} \tag{B.23}$$

$$= \frac{P(Y|Z,X)P(Z|X)}{P(Y|X)}, \tag{B.24}$$

which is the expected result.

References list

[5] T. Stoyanov et al. “Normal Distributions Transform Occupancy Map fusion: Simultaneous mapping and tracking in large scale dynamic environments”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 4702–4708.

[6] Jari Saarinen et al. “Fast 3D mapping in highly dynamic environments using normal distributions transform occupancy maps”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*. 2013, pp. 4694–4701.

[7] Claudia Stöcker et al. “Review of the Current State of UAV Regulations”. In: *Remote Sensing* 9.5 (2017). ISSN: 2072-4292.

[8] Frost & Sullivan. *Study analysing the current activities in the field of uav - Second Element: Way Forward*. Tech. rep. ENTR/2007/065. European Commission, 2007. URL: %5Curl%7Bhttps://ec.europa.eu/home-affairs/sites/homeaffairs/files/e-library/documents/policies/security/pdf/uav_study_element_2_en.pdf%7D.

[9] Alberto Vale, Rodrigo Ventura, and Paulo Carvalho. “Application of unmanned aerial vehicles for radiological inspection”. In: *Fusion Engineering and Design* 124 (2017). Proceedings of the 29th Symposium on Fusion Technology (SOFT-29) Prague, Czech Republic, September 5-9, 2016, pp. 492–495. ISSN: 0920-3796. URL: http://www.sciencedirect.com/science/article/pii/S0920379617307020.

[10] P.G. Martin et al. “The use of unmanned aerial systems for the mapping of legacy uranium mines”. In: *Journal of Environmental Radioactivity* 143 (2015), pp. 135–140. ISSN: 0265-931X. URL: http://www.sciencedirect.com/science/article/pii/S0265931X15000314.

[11] P.G. Martin et al. “3D unmanned aerial vehicle radiation mapping for assessing contaminant distribution and mobility”. In: *International Journal of Applied Earth Observation and Geoinformation* 52 (2016), pp. 12–19. ISSN: 0303-2434. URL: http://www.sciencedirect.com/science/article/pii/S0303243416300733.

[12] S. Dorafshan et al. “Challenges in bridge inspection using small unmanned aerial systems: Results and lessons learned”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. June 2017, pp. 1722–1730.

[13] Envision Intelligence. *Global Drones Market – Size, Outlook, Trends and Forecasts (2019 – 2025)*. Tech. rep. 2018.

[14] L. Canetta, G. Mattei, and A. Guanziroli. “Exploring commercial UAV market evolution from customer requirements elicitation to collaborative supply network management”. In: *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. June 2017, pp. 1016–1022.

[15] Terrence Fong and Charles Thorpe. “Vehicle Teleoperation Interfaces”. In: *Autonomous Robots* 11.1 (July 2001), pp. 9–18. ISSN: 1573-7527. URL: https://doi.org/10.1023/A:1011295826834.

[16] Olena Connor et al. “1. CERI Human Factors of UAVs: 2004 and 2005 Workshop Overviews”. In: *Human Factors of Remotely Operated Vehicles*. Emerald Group Publishing Limited, 2006. Chap. 5, pp. 3–20.

[17] R. T. Nullmeyer et al. “Birds of prey: Training solutions to human factors issues”. In: *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*. 2007.

[18] Robin R. Murphy. “Navigational and Mission Usability in Rescue Robots”. In: *Journal of the Robotics Society of Japan* 28.2 (2010), pp. 142–146.

[19] Commercial Aviation Safety Team and Flight Deck Automation Working Group. *Operational Use of Flight Path Management Systems. Final Report of the Performance-based operations Aviation Rulemaking Committee*. Tech. rep. Federal Aviation Administration, 2013. URL: https://www.faa.gov/aircraft/air_cert/design_approvals/human_factors/media/OUFPMS_Report.pdf.

[20] U.S. Department ofTransportation. *Office of Inspector General, Audit Report. ENHANCED FAA OVERSIGHT COULD REDUCE HAZARDS ASSOCIATED WITH INCREASED USE OF FLIGHT DECK AUTOMATION*. Tech. rep. AV-2016-013. Federal Aviation Administration, 2016. URL: https://www.oig.dot.gov/sites/default/files/FAA%20Flight%20Decek%20Automation_Final%20Report%5E1-7-16.pdf.

[21] Jari P. Saarinen et al. “3D normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments”. In: *The International Journal of Robotics Research* 32.14 (Sept. 2013), pp. 1627–1644.

[1] Hugo Courtois and Nabil Aouf. “Haptic feedback for obstacle avoidance applied to unmanned aerial vehicles”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2017, pp. 417–424.

[2] Hugo Courtois and Nabil Aouf. “Fusion of stereo and Lidar data for dense depth map computation”. In: *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*. IEEE. 2017, pp. 186–191.

[3] Hugo Courtois et al. “NDT RC: Normal Distribution Transform Occupancy Map with recentering”. Submitted to IEEE International Conference on Robotics and Automation.

[4] Hugo Courtois et al. “OAST: Obstacle Avoidance System for Teleoperation of UAVs”. Submitted to IEEE Transactions on Human-Machine Systems (THMS).

[22] Vincent Hayward et al. “Haptic interfaces and devices”. In: *Sensor Review* 24.1 (2004), pp. 16–29.

[23] S. Bouabdallah and R. Siegwart. “Full control of a quadrotor”. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2007, pp. 153–158.

[24] T. Lee, M. Leok, and N. H. McClamroch. “Geometric tracking control of a quadrotor UAV on SE(3)”. In: *49th IEEE Conference on Decision and Control (CDC)*. Dec. 2010, pp. 5420–5425.

[25] David Droeschel et al. “Multilayered Mapping and Navigation for Autonomous Micro Aerial Vehicles”. In: *Journal of Field Robotics* (2015).

[26] Sangyoon Lee et al. “Haptic control of a mobile robot: a user study”. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. 2002, 2867–2874 vol.3.

[27] T. M. Lam et al. “Artificial Force Field for Haptic Feedback in UAV Teleoperation”. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 39.6 (Nov. 2009), pp. 1316–1330. ISSN: 1083-4427.

[28] P. Stegagno et al. “A semi-autonomous UAV platform for indoor remote operation with visual and haptic feedback”. In: *2014 IEEE ICRA*. May 2014, pp. 3862–3869.

[29] A. M. Brandt and M. B. Colton. “Haptic collision avoidance for a remotely operated quadrotor UAV in indoor environments”. In: *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. Oct. 2010, pp. 2724–2731.

[30] Oussama Khatib. “Real time obstacle avoidance for manipulators and mobile robots”. In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[31] Thanh Mung Lam et al. “Force-Stiffness Feedback in Uninhabited Aerial Vehicle Teleoperation with Time Delay”. In: *Journal of Guidance, Control, and Dynamics* 32.3 (May 2009).

[32] R. M. Philbrick and M. B. Colton. “Effects of Haptic and 3D Audio Feedback on Operator Performance and Workload for Quadrotor UAVs in Indoor Environments”. In: *Journal of Robotics and Mechatronics* 26.5 (Feb. 2014), pp. 580–591.

[33] R. Mahony et al. “A new framework for force feedback teleoperation of robotic vehicles based on optical flow”. In: *Robotics and Automation, 2009. ICRA ’09. IEEE International Conference on*. May 2009, pp. 1079–1085.

[34] A. Kanso et al. “Enhanced teleoperation of UAVs with haptic feedback”. In: *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. July 2015, pp. 305–310.

[35] A. Rüesch et al. “Kinetic scrolling-based position mapping for haptic teleoperation of unmanned aerial vehicles”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. May 2012, pp. 3116–3121.

[36] Sammy Omari et al. “Bilateral Haptic Teleoperation of an Industrial Multirotor UAV”. In: *Gearing Up and Accelerating Cross-fertilization between Academic and Industrial Robotics Research in Europe*. Ed. by Florian Röhrbein, Germano Veiga, and Ciro Natale. Cham: Springer International Publishing, 2014, pp. 301–320. ISBN: 978-3-319-02934-4.

[37] Steven Martin and Nick Hillier. “Characterisation of the Novint Falcon Haptic Device for Application as a Robot Manipulator”. In: *Australasian Conference on Robotics and Automation (ACRA*. 2009.

[38] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA Workshop on Open Source Software*. Vol. 3. Jan. 2009, p. 5.

[39] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. Sept. 2004, 2149–2154 vol.3.

[40] Fadri Furrer et al. “RotorS - A Modular Gazebo MAV Simulator Framework”. In: *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, Feb. 2016, pp. 595–625. ISBN: 978-3-319-26054-9.

[41] S. G. Hart and L. E. Stavenland. “Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research”. In: *Advances in Psychology. Human Mental Workload*. Ed. by P. A. Hancock and N. Meshkati. Vol. 52. Elsevier, 1988. Chap. 7, pp. 139–183.

[42] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: `http://www.R-project.org/`.

[43] Douglas Bates et al. “Fitting Linear Mixed-Effects Models Using lme4”. In: *Journal of Statistical Software* 67.1 (Oct. 2015), pp. 1–48.

[44] Russell V. Lenth. “Least-Squares Means: The R Package lsmeans”. In: *Journal of Statistical Software* 69.1 (Jan. 2016), pp. 1–33.

[45] Sandra G. Hart. “Nasa-Task Load Index (NASA-TLX); 20 Years Later”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50.9 (Oct. 2006), pp. 904–908.

[46] Rebecca A. Grier. “How High is High? A Meta-Analysis of NASA-TLX Global Workload Scores”. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 59.1 (2015), pp. 1727–1731.

[47] Sebastian Thrun. “Exploring Artificial Intelligence in the New Millennium”. In: ed. by Gerhard Lakemeyer and Bernhard Nebel. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. Chap. Robotic Mapping: A Survey, pp. 1–35. ISBN: 1-55860-811-7.

[48] C. Cadena et al. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32.6 (Dec. 2016), pp. 1309–1332. ISSN: 1552-3098.

[49] F. Lu and E. Milios. “Globally Consistent Range Scan Alignment for Environment Mapping”. In: *Auton. Robots* 4.4 (Oct. 1997), pp. 333–349. ISSN: 0929-5593. URL: https://doi.org/10.1023/A:1008854305733.

[50] J. -. Gutmann and K. Konolige. “Incremental mapping of large cyclic environments”. In: *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA’99 (Cat. No.99EX375)*. Nov. 1999, pp. 318–325.

[51] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. “DTAM: Dense Tracking and Mapping in Real-time”. In: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2320–2327. ISBN: 978-1-4577-1101-5. URL: http://dx.doi.org/10.1109/ICCV.2011.6126513.

[52] Alberto Elfes. “Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation”. AAI9006205. PhD thesis. Pittsburgh, PA, USA, 1989.

[53] Hans P. Moravec. “Sensor Fusion in Certainty Grids for Mobile Robots”. In: *AI Magazine* 9 (June 1988), pp. 61–74.

[54] M. Herbert et al. “Terrain mapping for a roving planetary explorer”. In: *Proceedings, 1989 International Conference on Robotics and Automation*. May 1989, 997–1002 vol.2.

[55] R. Triebel, P. Pfaff, and W. Burgard. “Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2006, pp. 2276–2282.

[56] César Rivadeneyra and Mark Campbell. “Probabilistic Multi-level Maps from LIDAR Data”. In: *Int. J. Rob. Res.* 30.12 (Oct. 2011), pp. 1508–1526. ISSN: 0278-3649. URL: http://dx.doi.org/10.1177/0278364910392405.

[57] Armin Hornung et al. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013). Software available at http://octomap.github.com. URL: http://octomap.github.com.

[58] P. Biber and W. Strasser. “The normal distributions transform: a new approach to laser scan matching”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3. Oct. 2003, 2743–2748 vol.3.

[59] Martin Magnusson, Achim Lilienthal, and Tom Duckett. “Scan registration for autonomous mining vehicles using 3D-NDT”. In: *Journal of Field Robotics* 24.10 (), pp. 803–827.

[60] T. Stoyanov, M. Magnusson, and A. J. Lilienthal. “Point set registration through minimization of the L2 distance between 3D-NDT models”. In: *2012 IEEE International Conference on Robotics and Automation*. May 2012, pp. 5196–5201.

[61] Todor Stoyanov, Martin Magnusson, and Achim J. Lilienthal. “Comparative Evaluation of the Consistency of Three-dimensional Spatial Representations used in Autonomous Robot Navigation”. In: *Journal of Field Robotics* 30.2 (), pp. 216–236. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21446. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21446.

[62] D. Scaramuzza and F. Fraundorfer. “Visual Odometry: Part I - The First 30 Years and Fundamentals”. In: *IEEE Robotics Automation Magazine* 18.4 (Dec. 2011), pp. 80–92. ISSN: 1070-9932.

[63] F. Fraundorfer and D. Scaramuzza. “Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications”. In: *IEEE Robotics Automation Magazine* 19.2 (June 2012), pp. 78–90. ISSN: 1070-9932.

[64] Andreas Geiger, Julius Ziegler, and Christoph Stiller. “StereoScan: Dense 3D Reconstruction in Real-time”. In: *Intelligent Vehicles Symposium (IV)*. 2011.

[65] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. 2014, pp. 15–22.

[66] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct Sparse Odometry”. In: *CoRR* abs/1607.02565 (2016). arXiv: 1607.02565. URL: http://arxiv.org/abs/1607.02565.

[67] Axel Beauvisage, Nabil Aouf, and Hugo Courtois. “Multi-spectral visual odometry for unmanned air vehicles”. In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016, pp. 1994–1999.

[68] T. Mouats et al. “Multispectral Stereo Odometry”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.3 (June 2015), pp. 1210–1224. ISSN: 1524-9050.

[69] François Pomerleau, Francis Colas, and Roland Siegwart. “A Review of Point Cloud Registration Algorithms for Mobile Robotics”. In: *Found. Trends Robot* 4.1 (May 2015), pp. 1–104. ISSN: 1935-8253.

[70] P. J. Besl and N. D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 0162-8828.

[71] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. “Generalized-ICP”. In: *Robotics: Science and Systems*. Ed. by Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos. The MIT Press, 2009. ISBN: 978-0-262-51463-7.

[72] Martin Magnusson. “The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection”. PhD thesis. Örebro universitet, 2009.

[73] Y. Chen and G. Medioni. “Object modeling by registration of multiple range images”. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. Apr. 1991, 2724–2729 vol.3.

[74] Yang Chen and G. Medioni. “Object Modelling by Registration of Multiple Range Images”. In: *Image Vision Comput.* 10.3 (Apr. 1992), pp. 145–155. ISSN: 0262-8856.

[75] Todor Stoyanov et al. “Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations”. In: *The International Journal of Robotics Research* 31.12 (2012), pp. 1377–1393.

[76] Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In: *Commun. ACM* 18 (9 Sept. 1975), pp. 509–517. ISSN: 0001-0782. URL: http://doi.acm.org/10.1145/361002.361007.

[77] A. Censi. “An accurate closed-form estimate of ICP’s covariance”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Apr. 2007, pp. 3167–3172.

[78] B. Jian and B. C. Vemuri. “Robust Point Set Registration Using Gaussian Mixture Models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.8 (Aug. 2011), pp. 1633–1645. ISSN: 0162-8828.

[79] H. Hong and B. H. Lee. “Probabilistic normal distributions transform representation for accurate 3D point cloud registration”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 3333–3338.

[80] T. F. Chan, G. H. Golub, and R. J. LeVeque. “Updating Formulae and a Pairwise Algorithm for Computing Sample Variances”. In: *COMPSTAT 1982 5th Symposium held at Toulouse 1982*. Ed. by H. Caussinus, P. Ettinger, and R. Tomassone. Heidelberg: Physica-Verlag HD, 1982, pp. 30–41. ISBN: 978-3-642-51461-6.

[81] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[82] Manuel Yguel, Olivier Aycard, and Christian Laugier. “Update Policy of Dense Maps: Efficient Algorithms and Sparse Representation”. In: *Field and Service Robotics: Results of the 6th International Conference*. Ed. by Christian Laugier and Roland Siegwart. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 23–33. ISBN: 978-3-540-75404-6. URL: https://doi.org/10.1007/978-3-540-75404-6_3.

[83] J. Sturm et al. “A benchmark for the evaluation of RGB-D SLAM systems”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2012, pp. 573–580.

[84] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[85] Gaurav Pandey, James R Mcbride, and Ryan M Eustice. “Ford Campus Vision and Lidar Data Set”. In: *Int. J. Rob. Res.* 30.13 (Nov. 2011), pp. 1543–1552. ISSN: 0278-3649.

[86] Arun Das and Steven L. Waslander. “Scan registration using segmented region growing NDT”. In: *The International Journal of Robotics Research* 33.13 (2014), pp. 1645–1663.

[87] Amirhossein TAMJIDI and Cang YE. “6-DOF Pose Estimation of an Autonomous Car by Visual Feature Correspondence and Tracking”. In: *The In-*

ternational Journal of INTELLIGENT CONTROL AND SYSTEMS 17.3 (2012), pp. 94–101.

[88] Kruno Lenac et al. “Fast planar surface 3D SLAM using LIDAR”. In: *Robotics and Autonomous Systems* 92 (2017), pp. 197–220.

[89] Andreas Geiger, Martin Roser, and Raquel Urtasun. “Efficient Large-Scale Stereo Matching”. In: *Asian Conference on Computer Vision (ACCV)*. 2010.

[90] Rainer Kümmerle et al. “On Measuring the Accuracy of SLAM Algorithms”. In: *Auton. Robots* 27.4 (Nov. 2009), pp. 387–407. ISSN: 0929-5593. URL: http://dx.doi.org/10.1007/s10514-009-9155-6.

[91] Anestis Zaganidis et al. “Semantic-assisted 3D normal distributions transform for scan registration in environments with limited structure”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. 2017, pp. 4064–4069. URL: https://doi.org/10.1109/IROS.2017.8206262.

[92] S. M. Abbas and A. Muhammad. “Outdoor RGB-D SLAM Performance in Slow Mine Detection”. In: *ROBOTIK 2012; 7th German Conference on Robotics*. May 2012, pp. 1–6.

[93] Rostam Affendi Hamzah and Haidi Ibrahim. “Literature Survey on Stereo Vision Disparity Map Algorithms.” In: *J. Sensors* 2016 (2016), 8742920:1–8742920:23.

[94] *Velodyne VLP-16 LITE*. http://velodynelidar.com/vlp-16-lite.html. Accessed: 2017-05-05.

[95] Heiko Hirschmuller. “Stereo Processing by Semiglobal Matching and Mutual Information”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 30.2 (Feb. 2008), pp. 328–341. ISSN: 0162-8828.

[96] Hernán Badino, Daniel F. Huber, and Takeo Kanade. “Integrating LIDAR into Stereo for Fast and Improved Disparity Computation.” In: *3D Imaging, Modeling, Processing, Visualization, Transmission (3DIMPVT)*. IEEE Computer Society, 2011, pp. 405–412. ISBN: 978-0-7695-4369-7.

[97] Will Maddern and Paul Newman. “Real-time probabilistic fusion of sparse 3D LIDAR and dense stereo”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon, South Korea, Oct. 2016.

[98] Ondrej Miksik et al. “Incremental dense multi-modal 3D scene reconstruction”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. 2015, pp. 908–915.

[99] Henrik Andreasson, Rudolph Triebel, and Achim Lilienthal. “Non Iterative Vision Based Interpolation of 3D Laser Scans”. In: *Autonomous Robots and Agents*. Springer Berlin Heidelberg, 2007, pp. 83–90. ISBN: 978-3-540-73424-6.

[100] Cristiano Premebida et al. “High-resolution LIDAR-based Depth Mapping using Bilateral Filter”. In: *CoRR* abs/1606.05614 (2016).

[101] Moritz Menze and Andreas Geiger. “Object Scene Flow for Autonomous Vehicles”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[102] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0521540518.

[103] T. A. Clarke and J. G. Fryer. “The Development of Camera Calibration Methods and Models”. In: *The Photogrammetric Record* 16.91 (1998), pp. 51–66.

[104] R. Tsai. “A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses”. In: *IEEE Journal on Robotics and Automation* 3.4 (Aug. 1987), pp. 323–344. ISSN: 0882-4967.

[105] Zhengyou Zhang. “Flexible camera calibration by viewing a plane from unknown orientations”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 1. Sept. 1999, 666–673 vol.1.

[106] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[107] Jean-Yves Bouguet. *Camera Calibration Toolbox for Matlab*. 2015. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/ (visited on 08/12/2018).

[108] Yuri Boykov, Olga Veksler, and Ramin Zabih. “Fast Approximate Energy Minimization via Graph Cuts”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 23.11 (Nov. 2001), pp. 1222–1239. ISSN: 0162-8828. URL: https://doi.org/10.1109/34.969114.

[109] R. Spangenberg et al. “Large scale Semi-Global Matching on the CPU”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. June 2014, pp. 195–201.

[110] D. Hernandez-Juarez et al. “Embedded Real-time Stereo Estimation via Semi-global Matching on the GPU”. In: *Procedia Computer Science* 80 (2016). International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA, pp. 143–153. ISSN: 1877-0509.

[111] Richard O. Duda and Peter E. Hart. “Pattern Classification and scene analysis”. In: John Wiley and Sons, Inc., 1973, pp. 271–272.

[112] Irwin Sobel. “An Isotropic 3x3 Image Gradient Operator”. In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[113] C. Goerzen, Z. Kong, and B. Mettler. “A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance”. In: *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, U.S.A. June 8–10, 2009*. Ed. by Kimon P. Valavanis et al. Dordrecht: Springer Netherlands, 2010, pp. 65–100. ISBN: 978-90-481-8764-5. URL: https://doi.org/10.1007/978-90-481-8764-5_5.

[114] Voemir Kunchev et al. “Path Planning and Obstacle Avoidance for Autonomous Mobile Robots: A Review”. In: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. by Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 537–544.

[115] Anthony (Tony) Stentz. *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*. Tech. rep. CMU-RI-TR-93-20. Pittsburgh, PA: Carnegie Mellon University, Aug. 1993.

[116] Sven Koenig and Maxim Likhachev. “D*Lite”. In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 476–483. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=777092.777167.

[117] Liang Yang et al. “Survey of Robot 3D Path Planning Algorithms”. In: *J. Control Sci. Eng.* 2016 (Mar. 2016), pp. 5–. ISSN: 1687-5249. URL: https://doi.org/10.1155/2016/7426913.

[118] Ross E. Allen and Marco Pavone. “A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance”. In: *Robotics and Autonomous Systems* 115 (2019), pp. 174–193. ISSN: 0921-8890. URL: http://www.sciencedirect.com/science/article/pii/S0921889017308692.

[119] Jose Luis Sanchez-Lopez et al. “A Real-Time 3D Path Planning Solution for Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments”. In: *Journal of Intelligent & Robotic Systems* 93.1 (Feb. 2019), pp. 33–53. ISSN: 1573-0409. URL: https://doi.org/10.1007/s10846-018-0809-5.

[120] Michael Hoy, Alexey S. Matveev, and Andrey V. Savkin. “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey”. In: *Robotica* 33.3 (2015), pp. 463–497.

[121] J. Borenstein and Y. Koren. “High-speed obstacle avoidance for mobile robots”. In: *Proceedings IEEE International Symposium on Intelligent Control 1988*. Aug. 1988, pp. 382–384.

[122] J. Borenstein and Y. Koren. “The vector field histogram-fast obstacle avoidance for mobile robots”. In: *IEEE Transactions on Robotics and Automation* 7.3 (June 1991), pp. 278–288. ISSN: 1042-296X.

[123] I. Ulrich and J. Borenstein. “VFH+: reliable obstacle avoidance for fast mobile robots”. In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 2. May 1998, 1572–1577 vol.2.

[124] I. Ulrich and J. Borenstein. “VFH/sup */: local obstacle avoidance with look-ahead verification”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 3. Apr. 2000, 2505–2511 vol.3.

[125] Dong An and Hong Wang. “VPH: a new laser radar based obstacle avoidance method for intelligent mobile robots”. In: *Fifth World Congress on Intelligent Control and Automation (IEEE Cat. No.04EX788)*. Vol. 5. June 2004, 4681–4685 Vol.5.

[126] J. Gong et al. “VPH+: An Enhanced Vector Polar Histogram Method for Mobile Robot Obstacle Avoidance”. In: *2007 International Conference on Mechatronics and Automation*. Aug. 2007, pp. 2784–2788.

[127] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.

[128] S. Quinlan and O. Khatib. “Elastic bands: connecting path planning and control”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. May 1993, 802–807 vol.2.

[129] M. Zhang et al. “A high fidelity simulator for a quadrotor UAV using ROS and Gazebo”. In: *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*. Nov. 2015, pp. 002846–002851.

[130] I. P. Sary et al. “Design of Obstacle Avoidance System on Hexacopter Using Vector Field Histogram-Plus”. In: *2018 IEEE 8th International Conference on System Engineering and Technology (ICSET)*. Oct. 2018, pp. 18–23.

[131] Jungho Moon, Byung-Yoon Lee, and Min-Jea Tahk. “A Hybrid Dynamic Window Approach for Collision Avoidance of VTOL UAVs”. In: *International Journal of Aeronautical and Space Sciences* 19.4 (Dec. 2018), pp. 889–903. ISSN: 2093-2480. URL: https://doi.org/10.1007/s42405-018-0061-z.

[132] C. Berger et al. “Evaluation of reactive obstacle avoidance algorithms for a quadcopter”. In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. Nov. 2016, pp. 1–6.

[133] M. Hua and H. Rifaï. “Obstacle avoidance for teleoperated underactuated aerial vehicles using telemetric measurements”. In: *49th IEEE Conference on Decision and Control (CDC)*. Dec. 2010, pp. 262–267.

[134] X. Hou and R. Mahony. “Dynamic Kinesthetic Boundary for Haptic Teleoperation of VTOL Aerial Robots in Complex Environments”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46.5 (May 2016), pp. 694–705. ISSN: 2168-2216.

[135] M. Odelga, P. Stegagno, and H. H. Bülthoff. “Obstacle detection, tracking and avoidance for a teleoperated UAV”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 2984–2990.

[136] S. Bouabdallah and R. Siegwart. “Backstepping and Sliding-mode Techniques Applied to an Indoor Micro Quadrotor”. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Apr. 2005, pp. 2247–2252.

[137] John C. Hart. “Distance to an Ellipsoid”. In: *Graphics Gems II.1*. Ed. by Paul S. Heckbert. Academic Press, 1994, pp. 113–119. ISBN: 978-0-12-336156-1.

[138] Amit Reza and Anand S. Sengupta. “Least Square Ellipsoid Fitting Using Iterative Orthogonal Transformations”. In: *Appl. Math. Comput.* 314.C (Dec. 2017), pp. 349–359. ISSN: 0096-3003.

[139] D. Falanga, S. Kim, and D. Scaramuzza. “How Fast Is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid”. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 1884–1891. ISSN: 2377-3774.

[140] SC De Vries. *UAVs and control delays*. Tech. rep. TNO DEFENCE SECURITY and SAFETY SOESTERBERG (NETHERLANDS), 2005.

[141] W. R. Ferrell. “Remote manipulation with transmission delay”. In: *IEEE Transactions on Human Factors in Electronics* HFE-6.1 (Sept. 1965), pp. 24–32. ISSN: 2168-2852.

[142] T. B. Sheridan. “Space teleoperation through time delay: review and prognosis”. In: *IEEE Transactions on Robotics and Automation* 9.5 (Oct. 1993), pp. 592–606. ISSN: 2374-958X.

[143] Neal A. Tanner and Günter Niemeyer. “Improving Perception in Time-delayed Telerobotics”. In: *The International Journal of Robotics Research* 24.8 (2005), pp. 631–644.

[144] R. J. Anderson and M. W. Spong. “Bilateral control of teleoperators with time delay”. In: *IEEE Transactions on Automatic Control* 34.5 (May 1989), pp. 494–501. ISSN: 2334-3303.

[145] G. Niemeyer and J. -. E. Slotine. “Stable adaptive teleoperation”. In: *IEEE Journal of Oceanic Engineering* 16.1 (Jan. 1991), pp. 152–162. ISSN: 2373-7786.

[146] Günter Niemeyer and Jean-Jacques E. Slotine. “Telemanipulation with Time Delays”. In: *The International Journal of Robotics Research* 23.9 (2004), pp. 873–890.

[147] T. M. Lam, M. Mulder, and M. M. Van Paassen. “Haptic Feedback in Uninhabited Aerial Vehicle Teleoperation with Time Delay”. In: *Journal of Guidance, Control, and Dynamics* 31.6 (2008), pp. 1728–1739.